VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
INSTITUTE OF INFORMATICS

SPRÁVA ICT PROSTŘEDÍ POMOCÍ AUTONOMNÍCH ŘEŠENÍ
ADMINISTRATION OF ICT ENVIRONMENTS USING AUTONOMOUS SOLUTIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Filip Boštík

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Dydowicz, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Filip Boštík**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Informační management
Vedoucí práce:	**Ing. Petr Dydowicz, Ph.D.**
Akademický rok:	2016/17

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Správa ICT prostředí pomocí autonomních řešení

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrh řešení, přínos práce
Závěr
Seznam použité literatury

Cíle, kterých má být dosaženo:

Cílem této diplomové práce je návrh a tvorba automatického řešení pro vybraný problém správy operačních systémů serverů. Navržené řešení má sloužit ke snížení počtu rutinních zásahů prováděných administrátory na serverech s daným operačním systémem.

Základní literární prameny:

BASL, Josef a Roman BLAŽÍČEK. Podnikové informační systémy. Podnik v informační společnosti. 1. vyd. Praha: Grada, 2008. 283 s. ISBN 978-80-247-2279-5.

MOLNÁR, Zdeněk. Automatizované informační systémy. 1. vyd. Praha: Strojní fakulta ČVUT, 2000. 126 s. ISBN 80-01-02269-2.

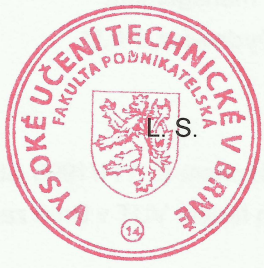
MOLNÁR, Zdeněk. Efektivnost informačních systémů. 1. vyd. Praha: Grada Publishing, 2000. 142 s. ISBN 80-7169-410-X.

ŘEPA, Václav. Analýza a návrh informačních systémů. 1. vyd. Praha: Ekopress, 1999. 403 s. ISBN 80-86119-13-0.

SODOMKA, Petr a Hana KLČOVÁ. Informační systémy v podnikové praxi. 2. aktualiz. a rozš. vyd. Brno: Computer Press, 2010. 501 s. ISBN 978-80-251-2878-7.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2016/17.

V Brně, dne 28. 2. 2017



L. S.

doc. RNDr. Bedřich Půža, CSc.
ředitel

doc. Ing. et Ing. Stanislav Škapa, Ph.D.
děkan

Abstrakt

Tato diplomová práce ukazuje příklad reaktivní automatizace v ICT prostředích. Je v ní popsáno řešení konkrétního problému operačního systému AIX, které je implementované v automatizačním nástroji IPC. Analýza současné situace zahrnuje rozbor vlastností operačního systému AIX, zkoumaného problému a rovněž nástroje IPC, který je použit pro implementaci řešení. Po popisu implementovaného řešení je provedeno rovněž ekonomické zhodnocení celé práce.

Abstract

This master's thesis shows an example of reactive server automation in ICT environments. There is a description of solution for specific problem in the AIX operating system which is implemented in an automation suite IPC. Analysis of current situation contains description of properties of the operating system AIX, the examined problem as well as of the IPC tool which is used for implementing this solution. After the description of the implemented solution there is also an economic evaluation of the entire thesis.

Klíčová slova

Administrace, ICT prostředí, automatické, řešení, problém, ticket, AIX, IPC.

Keywords

Administration, ICT environment, automatic, solution, problem, ticket, AIX, IPC.

BOŠTÍK, F. *Správa ICT prostředí pomocí autonomních řešení*. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2017. 78 s. Vedoucí diplomové práce Ing. Petr Dydowicz, Ph.D..

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 24. května 2017 .....

podpis studenta

Poděkování

Rád bych poděkoval lidem, bez kterých by tato práce nemohla vzniknout. Členům týmu Dynamic Automation, pro který jsem tuto práci tvořil, a hlavně mému manažerovi a mentorovi. Dále chci poděkovat odborníkům na operační systém AIX, se kterými jsem konzultoval návrh zde popisovaného programu. A nakonec vedoucímu této práce, panu Ing. Petru Dydowiczovi, Ph.D. za všechny rady, které mi během psaní této práce poskytl.

OBSAH

ÚVOD

Velké společnosti potřebují pro své fungování tisíce serverů. Na těchto serverech jsou nainstalované různé operační systémy a desítky různých aplikací, které dohromady tvoří IT infrastrukturu a informační systémy dané společnosti.

Se zvětšující se komplexitou těchto systémů ale zároveň roste i počet událostí a incidentů, které musí firmy řešit. Podniky proto buď rozšiřují stávající ICT oddělení, anebo volí metodu outsourcingu, kdy celou infrastrukturu (nebo její část) spravuje jiná firma. Ať již firma zvolí jeden nebo druhý způsob, tak starší metody správy informačních technologií přestávají stačit. Proto jsou tradiční řešení, jakým je například service desk, postupně nahrazována autonomními řešeními, která samostatně vyřeší většinu rutinních problémů a pomohou s analýzou a řešením i těch složitějších.

Mezi firmy, které tento problém při své činnosti řeší patří i IBM, která spravuje IT infrastrukturu stovkám zákazníkům po celém světě. A právě v oddělení, které se tímto problémem zabývá, píšu tuto práci, jejímž hlavním cílem je názorně ukázat, jak tato řešení fungují.

1 VYMEZENÍ PROBLÉMU A CÍLE PRÁCE

Cílem této práce je ukázat přínosy automatizace řešení rutinních problémů při správě serverů a aplikací společností na vybraném problému správy operačních systémů.

Problém, který v této práci řeším, je označován jako *'The largest dump device is too small'*. Jedná se o méně častý problém při správě operačního systému AIX. K řešení této chyby používám nástroj IPC. Jedná se o nástroj, který se ve společnosti IBM používá pro automatizaci správy ICT.

Dílčí cíle práce jsou:

- Přiblížit nástroj IPC a dané ICT prostředí.
- Charakterizovat operační systém AIX s důrazem na chybu *'The largest dump device is too small'* a její řešení.
- Popsat hlavní vlastnosti a fungování implementovaného programu.
- Ekonomicky zhodnotit přínos tohoto řešení.

Použité metody

Oddělení, pro které tuto práci píši, používá při vývoji upravenou metodiku SCRUM. Odchylkou od této metodiky je, že na konci sprintu není vylepšený program předán koncovému zákazníkovi, ale postup v návrhu a implementaci řešení je konzultován s experty na danou problematiku. Při každé iteraci zároveň dochází k testování nové a stávající funkcionality programu.

Další metodou, kterou jsem použil při návrhu popisovaného programu, je metoda expertních rozhovorů. Tyto rozhovory jsem opakovaně vedl s několika experty na OS AIX a nástroj IPC z řad seniorních administrátorů a vývojářů společnosti.

Jedním z dílčích cílů práce, je stanovení ekonomických přínosů implementovaného řešení. K tomu jsem používám metod Doby návratnosti investice a Return On Investment (ROI).

Řešený problém a nástroj IPC byly vybrány managementem vývojového oddělení, pro které tuto práci píši. Analýzy typu SWOT, SLEPTE nebo HOS, jsem tedy nevyužil, protože je nepovažuji za relevantní pro řešení daného problému.

2 TEORETICKÁ VÝCHODISKA PRÁCE

V této kapitole popíši znalosti, které by čtenář práce měl znát pro lepší pochopení této práce. Hlavní část kapitoly zabírá úvod do problematiky unixových systémů. Dále popíši několik důležitých pojmů, které se vztahují operačním systémům a programu, který budu navrhovat, jako je swap, JSON nebo regulární výrazy. Kapitolu zakončím úvodem do problematiky monitorování ICT infrastruktury.

2.1 Midrange počítače

Existuje několik možností, jak dělit počítače. Mezi ně patří rovněž dělení na mikropočítače (stolní počítače a laptopy), midrange (servery), mainframe (sálové počítače) a superpočítače (specializované, velmi výkonné počítačové clustery).

Jako 'midrange' jsou označovány počítače pro poskytování služeb (provoz firemních aplikací) zaměstnancům a klientům (většinou) malých až velkých podniků. Ve velkých podnicích a korporacích mohou tyto služby poskytovat větší mainframy. V dnešní době se namísto midrange používá častěji označení 'server'[1].

2.2 Middleware

Middleware může být definován jako SW, pro propojení dalších softwarových komponent (programů i aplikací). Hlavní úlohou middleware je poskytovat abstrakci při návrhu a vývoji SW. Příkladem midrange je AIS (Application Integratio SW). Jedná se o aplikace, jejichž úkolem je propojovat další aplikace (například informační systémy). Do Middleware se rovněž často řadí Java, které se budu věnovat v kapitole 2.6 [2].

2.3 Kernel

Kernel je názvem pro jádro operačního systému. Jedná se o program, který zajišťuje komunikaci operačního systému s hardwarem. Kernel si tedy vždy zachovává plnou kontrolu nad činností počítače. Mezi jeho hlavní činnosti patří správa procesů, paměti a vstupně-výstupních zařízení (pomocí ovladačů) [3].

Většina operačních systémů (nebo výrobců) má vlastní jádro. Kupříkladu:

- NT Kernel – pro Windows NT a novější (autor: Microsoft),
- Linux kernel – základ pro všechny linuxové distribuce (autor: Linus Torvalds),
- AIX kernel (autor: IBM) [4].

2.4 SWAP

Procesor během své činnosti ukládá právě zpracovávaná data (programy) do paměti. Ta má několik vrstev, přičemž platí, že čím blíže procesoru se daná paměť vyskytuje, tím větší je přístupová rychlost a kapacita klesá. Pro ukládání většiny zpracovávaných dat slouží operační paměť (RAM). Na disku je rovněž možné vyhradit část kapacity, kterou OS využije pro odkládání právě nevyužívaných dat, která již nelze uložit do operační paměti. Tomuto mechanismu se říká 'swapování' nebo 'paging' ('stránkování') [3].

2.5 Unixové systémy

Unixové systémy jsou operační systémy, které vycházejí z původního operačního systému UNIX, a které splňují standardy konsorcia The Open Group. Hlavními vlastnostmi těchto OS je, že umožňují současnou práci více uživatelům, kteří mohou současně pracovat s více programy (multi-user, multi-tasking system), a jsou nezávislé na konkrétní platformě (s některými výjimkami).

UNIX vznikl v roce 1969 v laboratořích Bell System, společnosti AT&T. Tvůrci původního systému byli Ken Thompson a Dennis Ritchie. V následujících letech byl UNIX licencován dalším společnostem a univerzitám. To vedlo ke vzniku variant původního OS, z nich některé jsou používané dodnes, např. BSD (Berkley), AIX (IBM) a HP-UX (HP). Kvůli složitým licenčním politikám spojených s Unix systémy, také začaly vznikat 'UNIX-like' systémy. Nejznámější z nich je Linux, který byl poprvé vydán roku 1991 (autorem je Linus Torvalds). Linux je kompletně samostatnou implementací, která s Unix nesdílí žádný zdrojový kód. Mezi další dnes rozšířené implementace systému patří OS X.

Operační systémy, které vycházejí z UNIX jsou buď proprietární (AIX, HP-UX) nebo Open-source (FreeBSD, Linux). Proprietárních implementace jsou hlavně používány na midrange počítačích. Open-source verze (hlavně Linux) poté nacházejí uplatnění nejen na midrange, ale také na síťových prvcích nebo osobních stanicích [5], [6], [7].

2.5.1 Základní vlastnosti

Největší výhodou Unix systémů je již zmíněná schopnost využívání jedné instance OS více uživateli, přičemž každý může používat více programů současně. Unixové OS mají většinou minimální nároky na Hardware, jedná se tedy o vysoce portabilní operační systémy. Výjimkou bývají proprietární verze systému, jako například AIX (který vyžaduje specifickou architekturu procesoru).

Pro Unixové systémy je vše soubor. To může pomoci ke snazší administraci systému (například pro nastavení nové instalace OS mnohdy stačí zkopírovat staré soubory s nastavením), nebo snazšímu hledání chyb (například ve příslušném logu, nebo souboru s nastavením).

Textové rozhraní systémů, tzv. 'terminál' (často je výchozí) uživateli umožňuje téměř jakoukoliv akci (pomocí vhodné kombinace jednoduchých příkazů). Nevýhodou terminálu ovšem je fakt, pro začínající uživatele není přívětivým rozhraním. Uživatel si sice může vyvolat nápovědu k téměř jakémukoliv příkazu, ale ta obvykle neobsahuje příklady použití. Příkazy dále mají krátká jména, která začínajícím uživatelům nemusí dávat smysl [1]. Systém rovněž provede jakoukoliv akci, kterou mu oprávněný uživatel zadá. Administrátor tedy kupříkladu může smazat celý systém jediným příkazem.

2.5.2 Fungování Unixových systémů

Všechny Unixové i UNIX-like systémy sdílí stejnou základní strukturu systému. Tu tvoří:

- Kernel – jádro systému, které tvoří vrstvu mezi HW a dalšími vrstvami systému (viz. kapitola 2.3).
- Příkazy (utility) – jsou malé programy, které uživatel nebo další aplikace mohou použít pro vlastní práci v systému, a uživatelé je spouštějí přes Shell (viz. kapitola 2.5.5).
- Shell – je interpretem příkazů uživatele (viz. kapitola 2.5.6) [6].

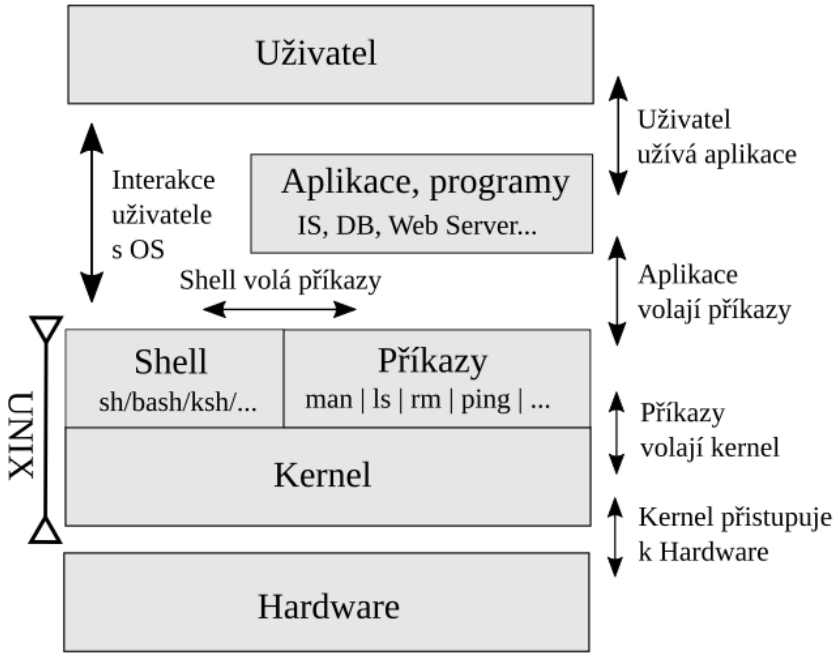
Tyto 3 vrstvy jsou sice základem, na kterém Unix pracuje, ale Unixové systémy samozřejmě podporují také složitější programy a aplikace, které uživatelé potřebují provozovat (například IS).

Unix byl postavený na několika základních principech, které určují způsob jeho používání. Podle těchto principů je lepší psát jednoduché programy, které jsou navržené pouze pro jednu činnost, než komplikované víceúčelové aplikace. Programy by měly být psané tak, aby mohly fungovat společně, a jejich vstup i výstup má proto být čistě textový.

Unixové systémy pracují s jakýmkoliv zdrojem (programy, daty, diskovými oddíly) vždy jako se souborem. Až způsob interakce s tímto souborem určí, jak s ním systém bude pracovat.

Praktickým výsledkem zmíněných principů je, že Unixové systémy nepotřebují grafické uživatelské rozhraní (GUI), a interakce s nimi může být čistě textová. Uživatel při práci se systémem (přesněji řečeno shellem) kombinuje jednoduché příkazy tak, aby dosáhl svého

[1] Třeba 'rm' - remove (odstraň soubor) nebo 'mkfs' - make filesystem (vytvoř souborový systém).



Obrázek č. 2.1: Princip fungování Unixových systémů [Vlastní zpracování]

cíle. Shell poté příkazy uživatele provede a okamžitě poskytuje výsledky, opět v textové formě.

Pokud uživatel chce mít na svém systému GUI, tak jej může doinstalovat. Uživatelské rozhraní poté běží jako další program, který je možné kdykoli zastavit nebo restartovat (bez nutnosti restartovat OS), popřípadě také vyměnit za jiné [6], [7], [8].

2.5.3 The Open Group

The Open Group je technickým konsorciem, které tvoří standardy pro vývoj open source technologií a certifikací podle těchto standardů. Jednou z těchto technologií je UNIX. Mezi nejdůležitější (platinové) členy konsorcia se řadí společnosti jako Fujitsu, HP, IBM nebo Philips[5].

2.5.4 POSIX

Termín ’POSIX’ je zkratkou pro ’Portable Operating System Interface’. Jedná se o několik IEEE standardů, které definují základní služby potřebné pro návrh aplikací, které bude možné snadno přenášet mezi různými operačními systémy.

POSIX tedy prakticky definuje funkcionalitu příkazů a shellu na Unixových operačních systémech tak, aby tyto systémy byly vzájemně kompatibilní [9].

2.5.5 Příkazy

Jako příkazy[2] jsou v Unixových systémech označovány malé programy, které jsou dostupné v základním OS a uživatelem jsou přímo volané přes shell. Příkazy mají (podle základní filozofie systému) pouze jediný účel, a jejich výstup je modifikovatelný několika parametry. Unixové systémy obsahují stovky příkazů, od příkazů pro editování nebo vyhledávání v textu (nebo textovém výstupu jiných příkazů), přes příkazy pro práci se soubory, až po příkazy, které mohou ovlivňovat fungování samotného OS [6], [7].

Pro lepší pochopení uvedu několik často používaných příkazů:

- *echo* – vypíše zadaný text do terminálu,
- *grep* – z textového vstupu vypíše pouze řádky, které splňují daná kritéria,
- *mv* – přesune soubory,
- *cat* – vypíše obsah souboru,
- *sed, awk* – příkazy, které umožňují pokročilé úpravy textu [7].

Parametry příkazů mohou mít také velkou roli pro výstup konkrétního příkazu a lze je navíc kombinovat. Před parametry se píše pomlčka. Jako ukázku použiji příkaz *ls*, který vypíše obsah adresáře. Jeho parametr *-a* vypíše všechny soubory (i skryté). Parametr *-l* změní výpis ze seznamu souborů na detailní řádkový výpis. Užití parametru *-R* poté vede k rekurzivnímu výpisu (tedy výpisu vnořených adresářů) [7].

2.5.6 Shell

Shell je textovým rozhraním, které slouží k překladu příkazů uživatele. Uživatel tedy na pokyn systému zapíše pokyny, které má systémy následovat, a to pomocí příkazů a několika zvláštních znaků, které shell následně přeloží a provede. Uživatel může rovněž vytvořit soubor (shellový skript), který po spuštění provede více příkazů.

Příkazy lze rovněž slučovat dohromady pomocí vybraných zvláštních znaků, mezi které patří například: '|', '>' nebo ';'. Znak '|' (anglicky označovaný jako *pipe*, česky jako *roura*), slouží ke spojení dvou příkazů. Systém poté spustí oba příkazy, a výstup prvního posílá řádek po řádku do druhého příkazu. Druhý příkaz poté upravený řádek vypíše [7].

```
ls -la | grep *.pdf
```

Výpis č. 2.1: Příkaz pro vypsání PDF souborů v adresáři [Vlastní zpracování]

Pojem shell rovněž slouží jako obecnější označení pro více implementací, které se od sebe mohou zásadně lišit. Mezi ně patří:

[2] V anglickém jazyce označované jako *utilities* nebo *commands*.

- *sh* – Bourne shell,
- *bash* – Bourne Again shell,
- *ksh* – Korn shell [6].

2.5.7 Root a sudo

V každém OS založeném na Unix je ve výchozím stavu vytvořen uživatel *root*. Jedná se o administrátorský účet, který má v systému všechna práva, včetně správy ostatních uživatelů. Root, někdy zvaný *superuser* rovněž může svá práva zapůjčit ostatním uživatelům. Jednou z možností, jak toho docílit, je použití příkazu *'su -'*, po kterém musí uživatel zadat heslo pro root. To jej přihlásí jako uživatele root. Z bezpečnostních důvodů (sdíleného hesla) se ovšem této možnosti příliš nepoužívá.

Druhou možností používání zvýšených práv, je tzv. *sudo* (z anglického 'superuser do'). Sudo umožňuje právě přihlášenému uživateli spouštět příkazy, které by byly jinak dostupné pouze administrátorovi. Root může měnit seznam příkazů, které mohou uživatelé použít s příkazem sudo. Sudo se používá tak, že před vybrané příkazy přidáme klíčové slovo *sudo* [3].

2.5.8 Proces a démon

Programy v Unixových OS se nazývají procesy. Jejich základní dělení je na procesy a démony.

Proces je tedy obecnějším označením pro program. Pokud je program spuštěn uživatelem, tak se jedná o tzv. 'foreground process'. Opakem je poté 'background process', který běží nezávisle na uživateli [3] .

Démon je druhem procesu běží nepřetržitě na pozadí systému, a čeká na situaci, pro kterou byl vytvořen. Příkladem může být *sendmail*, který odesílá emaily. Změna stavu démona je možná jen přes root[10].

2.5.9 Balíčky

Instalace nového SW je na Unixovém systému možná dvěma (dnes) běžnými způsoby. Jednak přímým spuštěním instalačního souboru (tzv. balíčku) nebo pomocí *package manager* (správce balíčků). Tyto postupy nejsou napříč platformami standardizované, ale jejich principy si jsou podobné. Instalace SW může být rovněž možná pomocí dedikovaného programu (například shell skriptu).

[3] Proces v popředí a proces na pozadí

Při přímé instalaci z (již staženého) balíčku může uživatel použít systémového příkazu. *Package manager* většinou umožňuje provést instalaci balíčku včetně stažení z dedikovaného uložiště (tzv. repositáře). Dále umí provést update balíčků nebo jejich odinstalování. Příklady jsou příkazy *rpm* a *dnf* na Red Hat Enterprise Linux [11].

2.5.10 SSH

SSH (neboli *Secure Shell*) je protokol, který zajišťuje plně šifrovanou komunikaci mezi počítači. Jeho tradičním (ale ne jediným) použitím je vzdálená administrace operačního systému. Přihlášení (autentizace) do vzdáleného počítače je možná dvěma způsoby: uživatelským jménem a heslem nebo pomocí kryptografických klíčů. Druhá možnost umožňuje uživateli přímé přihlášení do počítače bez nutnosti zadávat heslo. Vzdálený počítač již ale musí obsahovat veřejný klíč uživatele. Obdobnou funkcionalitu umožňuje v počítačích s Windows *PowerShell* (obdoba shellu v systému Windows) [7].

2.6 Java

Java je programovacím jazykem, který je používán převážně pro tvorbu větších firemních, webových a mobilních aplikací. Dříve byla Java rovněž používána pro tvorbu appletů. Applet je interaktivní aplikace běžící ve webovém prohlížeči. Dnes se ale od Javy v tomto ohledu ustupuje.

Hlavní vlastností Javy je portabilita naprogramovaného řešení. Kód v Javě ke svému běhu totiž využívá tzv. *Java Virtual Machine* (JVM), který funguje jako interpreter na daném operačním systému. JVM tedy funguje jako jednotné prostředí, které překládá kód Javy pro daný OS. Java je tudíž interpretovaný jazyk. Další důležitou vlastní Javy je její objektová orientace. Pro jazyk Java je totiž vše objekt [12].

2.7 JavaScript

JavaScript (JS) je objektově orientovaný interpretovaný programovací jazyk. Od svého vzniku v roce 1995 byl JavaScript používán hlavně ve webových prohlížečích pro zvýšení interaktivity webových stránek. V dnešní době tento jazyk získává na popularitě, a používá se i pro celou řadu dalších úkolů - od serverové části webových aplikací (*Node.js*) přes skriptování ve větších aplikacích, až po desktopové aplikace (knihovna *Electron*). JS (někdy označovaná jako *ECMAScript*) není odvozený od jazyka Java [4], ale Java obsahuje běhové prostředí pro JavaScript [13].

[4] Výběr jména byl marketingový tah.

2.7.1 JSON

JSON je zkratkou pro *JavaScript Object Notation*. Jedná se o způsob, jakým může programátor uložit JavaScriptový objekt (anebo libovolná data) do řetězce (textové proměnné). JSON funguje podobně jako pole, ale jednotlivé hodnoty jsou ukládány do párů *klíč* - *hodnota*, kde *klíč* je textový identifikátor záznamu a *hodnota* může obsahovat prakticky libovolný datový typ. JSON dnes není využívaný jen v JS, ale je podporován ve všech moderních programovacích jazycích, a je často používán namísto XML pro předávání dat mezi aplikacemi [13].

```
{
    "jméno_prace": "Správa ICT prostředí pomocí\
        autonomních řešení",
    "autor": "Bc. Filip Boštík",
    "kapitoly": [
        {
            "nazev_kapitoly": "Úvod",
            "prvni_strana_kapitoly": 6
        },
        {
            "nazev_kapitoly": "Teoretická východiska",
            "prvni_strana_kapitoly": 10
        }
    ]
}
```

Výpis č. 2.2: Ukázka zápisu JSON [Vlastní zpracování]

2.8 Regulární výrazy

Regulární výraz (často zkracované na *RegExp* nebo *RegEx*) je způsob popisu vzorů v textových řetězcích. Regulární výraz umožňuje ověřit, zda-li text odpovídá definovanému vzoru, anebo z tohoto textu extrahovat určité části. RegExp není specifický pro určitý jazyk, a proto různé implementace existují v mnoha programovacích jazycích (JavaScript nevyjímaje).

Regulární výraz (v JS) se skládá ze znaků uspořádaných dle zvláštních pravidel, obvykle uzavřených v lomítkách. Kód 2.3 obsahuje RegExp pro ověření správnosti emailové adresy. Tento regulární výraz by zachytil veškeré emaily, které by začínaly libovolnými tisknutelnými znaky (a tuto část, uživatelské jméno, by uložit). Po znaku zavináč by email

musel pokračovat další libovolně dlouhou sérií tisknutelných znaků. Konec emailu by musela být tečka následovaná jedním až třemi písmeny [13].

```
/(\S+)@\S+\.\w{1,3}/
```

Výpis č. 2.3: Regulární výraz pro ověření emailové adresy [Vlastní zpracování]

2.9 Událost a incident

Událost je ve správě informačních technologií chápána jako změna v (ICT) systému, která má význam pro management daného systému.

Incident značí událost, která ovlivnila fungování (ICT) systémů podniku.

Problém je zdrojem incidentů, a v době záznamu problému obvykle není znám [14].

2.10 Logování událostí v operačních systémech a aplikacích

V každém software se vyskytují odchylky od požadovaného stavu. Tyto odchylky mohou zahrnovat vše od informačních hlášení o stavu aplikace, přes varování až po závažné chyby. Všechna tato hlášení je třeba zaznamenávat a po určitou dobu ukládat. Operační systémy proto mají vestavěné standardizované způsoby, které umožňují tato hlášení ukládat (například *syslog*). Programovací jazyky obsahují vestavěné moduly pro zápis do systémových (nebo vlastních) logů.

Seznam údajů standardně ukládaných do logů se dělí do základních kategorií:

- Kdy – čas a datum události anebo zápisu.
- Kde – identifikátory aplikace, služeb, popřípadě části SW, která událost vyvolala (například knihovna nebo modul).
- Kdo – identifikátor uživatele, který daný SW spustil (popřípadě identifikátor OS).
- Co – popis a klasifikace (označení) hlášení [15].

2.11 Správa služeb ICT a monitorovací systémy

Dnešní firmy potřebují pro své fungování stovky, někdy i tisíce serverů - ať fyzických nebo virtuálních. Z praktických důvodů tedy není možné, aby byl každý server neustále přímo monitorován administrátorem. Proto dnešní společnosti využívají nejrůznější monitorovací řešení, která pomáhají s managementem těchto zpráv.

2.11.1 Funkce správy služeb IT podle ITIL

IT Infrastructure Library rozděluje správu služeb IT do následujících logických celků:

- Service Desk – je kontaktním místem zákazníků služby (ať už zaměstnanců firmy, nebo externích klientů).
- Technická správa – odpovídá za správu infrastruktury IT, se zaměřením na školení uživatelů a zaměstnanců firmy, řízení provozu IT a vylepšování stávající infrastruktury.
- Správa provozu IT – má za hlavní cíl správu fyzické infrastruktury IT.
- Správa aplikací – odpovídá za správu aplikací svých zákazníků (které jsou technickou implementací obchodních procesů) [16].

2.11.2 Možná praktická implementace Service Desk

Firmy často při provozu IT dělí správu IT na základě definice ITIL. Podle ITIL tedy mají service desky, které monitorují IT infrastrukturu nebo aplikace zákazníka. Tato oddělení mohou být rozdělena podle zkušeností zaměstnanců. Service desk (tzv. *1st level*) může přijímat požadavky zákazníků a provádět monitorování stavu serverů (aplikací). Případný problém může vyřešit podle definovaných postupů anebo jej posunout na vyšší úroveň správy systémů – do týmů technické správy, správy provozu IT nebo správy aplikací. Tyto vyšší úrovně správy (často ještě dále dělené na *2nd level* a *3rd level*) jsou tedy více specializované (dle operačních systémů, nebo konkrétních aplikací). V praxi platí, že vyšší úroveň service desků se zabývá složitějšími technickými problémy [17].

2.12 Monitorovací systémy

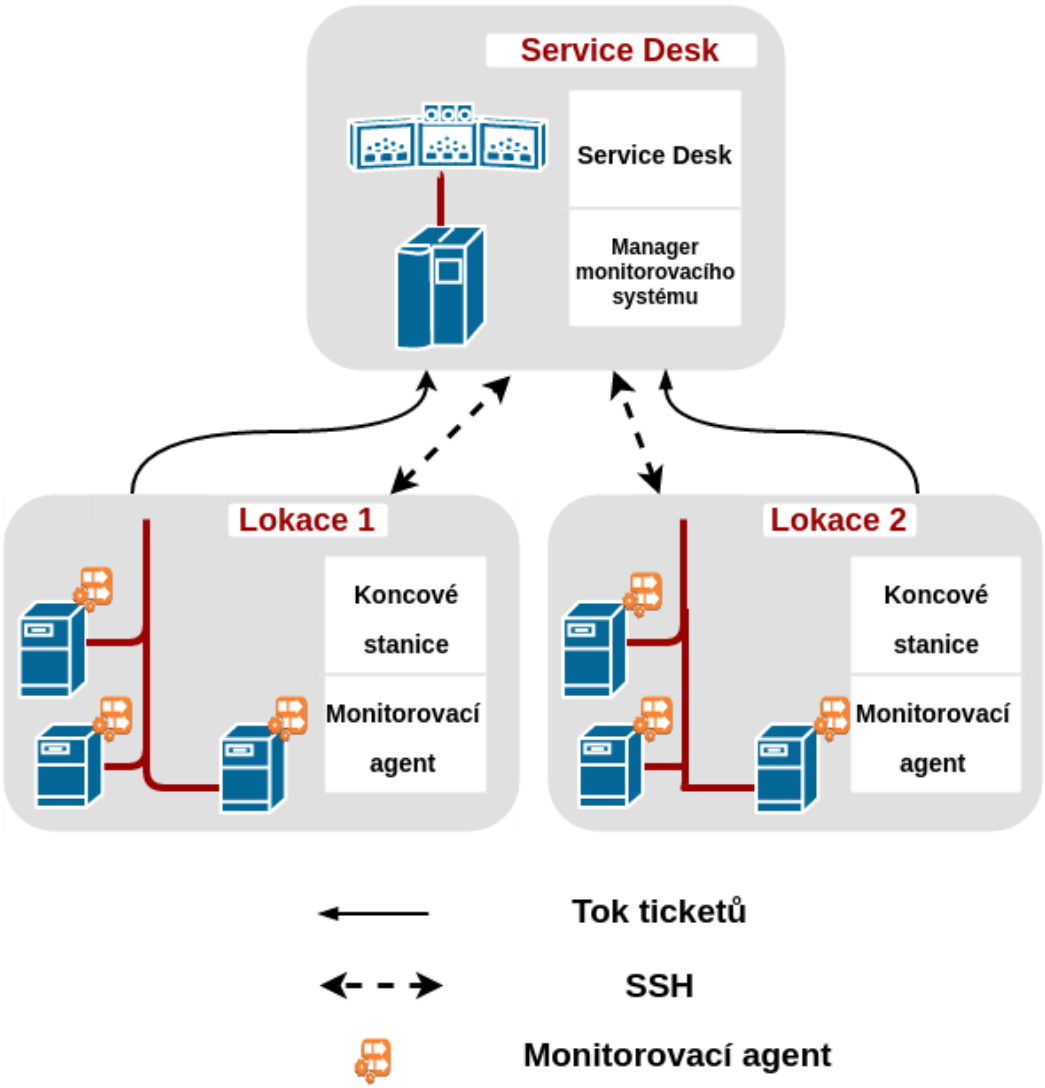
Pro sledování stavu serverů se používá tzv. *monitorovací software*. Jeho úkolem je pravidelná kontrola stavu serveru a hlášení případných problémů pracovníkům IT oddělení.

Řešení pro monitorování lze dělit podle zaměření řešení:

- *Server Monitoring SW* – může sledovat veškeré dění na serveru, tedy nejen OS, ale také chyb aplikací hlášených standartními způsoby.
- *Application Monitoring SW* – jsou zaměřené na konkrétní aplikace.
- *Network Monitoring SW* – sleduje aktuální stav počítačových sítí [17].

Dalším důležitým aspektem při výběru monitorovacího řešení, je rovněž jeho implementace. Ta může být buď *s agentem* nebo *bez agenta*. Společným prvkem všech řešení ale je, že mají společný centrální uzel (tzv. *manager*), který zpracovává aktuální stav IT prostředí.

- *Řešení manager-agent* – fungují tak, že na serveru je nainstalován zvláštní program, jehož cílem je pravidelné odesílání stavu *managerovi*, nebo aktualizace stavu na vyžádání. Další rolí agenta může být přístup na server (pro administrátora nebo automatizační platformu).
- *Řešení bez agenta* – nevyžaduje instalaci dedikované aplikace na koncový uzel. Kontrola aktuálního stavu serveru je poté zcela na *managerovi*, který se na server připojuje kupříkladu pomocí SSH. Manager tedy musí mít přístupové údaje na všechny spravované servery [17].



Obrázek č. 2.2: Monitorovací systémy manager-agent [Vlastní zpracování]

Existuje celá řada *monitorovacích systémů*, od jednoúčelových systémů pro malé firmy (jako správa síťové infrastruktury), po víceúčelové systémy navržené pro velké korporace, které mohou pomáhat spravovat aktiva celé firmy (ty jsou spíše označovány jako EAM – Enterprise Asset Management). Jako příklady těchto systémů bych uvedl Nagios

XI (od Nagios Enterprises), N-central (SolarWinds), PRTG Network Monitor (Paessler) nebo Maximo Asset Management (IBM) [18].

Ticket

Jako 'ticket' je v IT označována zpráva o *incidentu* nebo *service request* (žádosti o službu). Tickety bývají většinou zpracovávány v softwarech typu *Issue Tracking*, které mohou být napojeny na Monitorovací systémy[5]. Ticket se většinou týká jedné události nebo incidentu a tato zpráva poté obsahuje údaje jako stručný popis problému, přiřazení odpovědnosti zaměstnanci a seznam činností provedených s daným ticketem (log). Dalšími údaji může být kategorizace události, nebo zprávy zaměstnanců, kteří na ticketu pracovali [17].

S ticketem lze provést akce jako:

- *Diagnosis* – diagnóza je prozkoumáním dané události a hledáním příčiny.
- *Remediation* – remediace je vyřešením dané události nebo incidentu a následné uzavření ticketu.
- *Escalation* – eskalace značí předání odpovědnosti na vyšší úroveň funkce správy služeb IT.
- *Close* – zavření ticketu z jakéhokoliv důvodu [17].

2.13 Reakce na události ve správě služeb ICT

Při monitorování ICT prostředí lze definovat tři základní přístupy, podle času reakce na událost:

- *Reaktivní* – reakce na již vzniklý incident v systému.
- *Proaktivní* – reaguje na stav systému, který je v rozporu s požadovaným stavem, ale ještě nezpůsobil událost.
- *Prediktivní* – využívá analýzy daného prostředí k předcházení možných budoucích problémů [19].

Pro pochopení těchto přístupů použiji příklad *CPU High* – plně vytíženého procesoru. Reaktivní reakce na CPU High začíná ve chvíli, kdy server zaznamenal vysoké zatížení procesoru, a vytvořil ticket. Na ticket poté zareaguje operátor, který se na server připojí, a problém se pokusí vyřešit – například vypnutím některých procesů.

Proaktivní zásah proti CPU High může začít rutinním skenováním systému, které odhalí nechtěný stav systému – například starší verzi programu, která má horší optimalizaci. Proaktivním zásahem by poté byl update programu na novější verzi.

[5] Nebo se může jednat o jeden SW s vlastnostmi obou druhů systémů.

Při dlouhodobější analýze daného prostředí mohou být zjištěny příznaky budoucí zvýšené zátěže na procesor. Systém je poté nastaven tak, aby určité procesy pozastavil, pokud detekuje dříve zjištěné náznaky. Jedná se tedy o prediktivní zásah.

3 ANALÝZA PROBLÉMU A SOUČASNÉ SITUACE

Nyní se budu věnovat popisu problému, který budu řešit ve čtvrté kapitole. Náplní první části této kapitoly, bude stručný popis daného ICT prostředí, ve kterém je moje řešení nasazeno. Poté se budu zabývat popisu operačního systému AIX, a rozboru řešeného problému 'The largest dump device is too small.' Celou kapitolu následně zakončím popisem použitého nástroje IPC, což je nástroj pro automatizaci správy ICT prostředí.

3.1 Zkoumané prostředí

Zkoumané prostředí je tvořeno infrastrukturou všech zákaznických organizací sídlících v Evropě, které si nechávají spravovat IT infrastrukturou firmou IBM. Obsahuje desítky tisíc zákaznických serverů, které jsou spravovány desítkami service desků. Servery mají různé operační systémy, včetně různých verzí Windows, Linux, AIX nebo HP-UX. Zákaznické infrastruktury rovněž používají několik různých monitorovacích nástrojů, jako je IBM Maximo nebo Nagios. Tato infrastruktura je rozdělena mezi několik instancí nástroje IPC, které pomáhají se správou těchto serverů.

3.2 Operační systém AIX

AIX je proprietární operační systém společnosti IBM, zacílený na provoz klíčových firemních aplikací. AIX (Advanced Interactive eXecutive) patří mezi unixové systémy, konkrétně je založený na UNIX System 5 a BSD. Tento operační systém splňuje standardy POSIX a je certifikovaný konsorciem The Open Group. První verze systému byla uvedena na trh v roce 1986. Nejnovější verzí (k lednu 2017) je AIX 7.2. AIX mohl být provozován na několika platformách, v současné době je ale téměř výhradně provozován na systémech Power, pro které je optimalizovaný [20], [21].

3.2.1 Klíčové vlastnosti systému

Jak již bylo řečeno, AIX je zaměřený na provoz klíčových podnikových aplikací, a proto je optimalizován pro co nejvyšší bezpečnost systému, maximální využití výpočetní síly HW a minimalizaci času, kdy server, potažmo provozované aplikace, nemohou být v provozu (downtime). Následuje popis některých důležitých vlastností systému.

AIX binary compatibility guarantee[1] je zárukou IBM, že aplikace splňující běžné programovací standardy poběží na novějších verzích OS AIX bez rekompilace nebo modifikací.

[1] Záruka binární kompatibility

V případě, že by aplikace nefungovaly, tak se IBM zavazuje pomoci s vyřešením problému.

The Live Update je funkcionalitou přidanou ve verzi 7.2, kdy po instalaci updatů jádra není nutné restartovat systém.

PowerVM Live Partition Mobility je technologií, která umožňuje přesun právě používaného diskového oddílu mezi servery, aniž by bylo nutné běžící aplikace vypnout.

Systémy AIX obsahují řadu technologií pro podporu virtualizace, mimo jiné Workload Partitions (WPARs), Active Memory Expansion, PowerVM hypervisor. *PowerVM hypervisor* je vestavěn přímo do firmwaru procesorů systémů Power a umožňuje rozdělení výpočetní síly těchto procesorů pro jednotlivé instance operačních systémů. Ať už statickým přidělením priority, anebo dynamicky podle aktuálního vytížení jednotlivých systémů. *Workload Partitions*[2] umožňují rozdělení systémových zdrojů pro aplikace (nebo vývojáře) tak, aby každá aplikace běžela v samostatném obrazu operačního systému. *Active Memory Expansion*[3] komprimuje obsah operační paměti za běhu a rozšiřuje tak RAM o desítky procent.

Trusted Execution je jedním z bezpečnostních prvků, kdy je povoleno spouštění pouze důvěryhodných (nepozměněných) aplikací.

V oblasti správy systému AIX nabízí *smitty* - textové rozhraní, které administrátorům umožňuje provést téměř veškerou administraci systému. AIX dále obsahuje sadu příkazů pro LVM - *Logical Volume Management*.

AIX rovněž na Power Systems podporuje užití tzv. *LPAR (Logical Partitions)*, což umožňuje běh více instancí OS (AIX a Linux) na jednom fyzickém serveru. Tím lze dosáhnout maximálního využití výkonu HW.

AIX oficiálně podporuje open source a GNU nástroje, například přes *AIX Toolbox for Linux Applications*. Jedná se o soubor původně Linuxových aplikací (balíčků), které je možné použít i na AIX [20], [22], [23].

3.2.2 HW nároky

AIX byl v minulosti používán na několika platformách, vyvíjených firmou IBM. V současnosti je však podporována pouze architektura POWER, pro kterou je OS navržen [21].

[2] Pracovní oddíly

[3] Aktivní rozšíření paměti

IBM Power Systems

Power Systems je označení řady serverů společnosti IBM, jejichž architektura je založená na procesorech POWER[4]. Do roku 2008 byly označované jako *System p* (předtím *RS/6000*).

První procesory řady POWER vznikly na přelomu 80. a 90. let minulého století, jakožto výsledek vývoje CPU s architekturou RISC. V současné době tyto procesory podporují operační systémy AIX, *IBM i* a 64bitové verze Linux, přičemž poskytují HW podporu pro mnoho klíčových vlastností, které AIX nabízí (PowerVM, běh více instancí OS současně…) [21].

3.2.3 Verze

Při práci s jakýmkoliv operačním systémem je nutné pochopit značení jednotlivých verzí. Označení OS AIX se skládá z verze systému (například AIX 7.1), Technické úrovně a Servisních balíků. Verzi systému je tedy možné zapsat buď plným zápisem (AIX 7.1 TL3 SP2) anebo jako jedno číslo (např.: 6100-00-02-0750).

Hlavní verze[5] systému jsou v současné době podporovány 10 let (podporu poslední technické úrovně lze rozšířit). Tyto verze se od sebe mohou značně lišit, protože každá nová verze obsahuje nové technologie.

Technické úrovně (Technical Level nebo TL) přinášejí další rozšíření funkcionality. Vycházejí častěji, obvykle mezi nimi není odstup větší jak dva roky.

Servisní balík (Service Pack - SP) slouží k instalaci záplat a rozšíření podpory pro servery (HW). SP vycházejí dvakrát, někdy i třikrát za rok [20].

Prozatimní opravy (Interim Fixes) obsahují opravy kritické pro správný chod systému.

Vybraná nová funkcionalita hlavních verzí AIX:

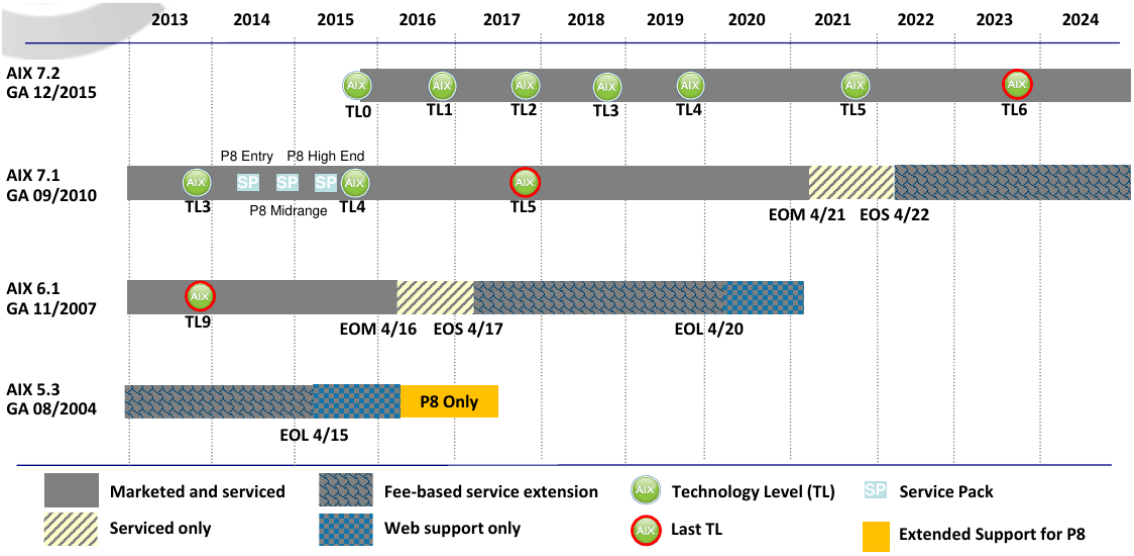
- AIX 5.1 (představen: 2001) – *Linux Affinity* - podpora linuxových aplikací.
- AIX 5.2 (2002)
- AIX 5.3 (2004) – vylepšení funkcionality *dump*.
- AIX 6.1 (2007) – Workload Partitions, Live Partition Mobility, Trusted Execution…
- AIX 7.1 (2010) – podpora až 256 jader procesoru (1024 vláken) jedním LPAR.
- AIX 7.2 (2015) – možnost aplikace prozatímních oprav systému (updatu kernelu) bez nutnosti restartu systému [24].

[4] *POWER* je zkratkou pro *Power Optimization with Enhanced RISC.*

[5] *Major version*, někdy též Base level (Základní úroveň).

Užívané verze systému

AIX byl uveden na trh v roce 1986. Nejnovější verzí je AIX 7.2. Obrázek 3.1 ukazuje životní cyklus podporovaných verzí.



Obrázek č. 3.1: Životní cyklus podporovaných verzí AIX [25]

Z obrázku vyplývá, že plně podporovaná je verze 7.2 a 7.1, a verze 6.1 a 5.3 jsou podporované již jen v rámci rozšířené podpory To ovšem neznamená, že tyto nebo starší verze nejsou vůbec používány. Tabulka 3.1[6] ukazuje rozložení jednotlivých verzí systému AIX, které jsou zastoupené ve zkoumaném ICT prostředí. V tomto prostředí se v současné době nachází více než tři tisíce serverů s operačním systémem AIX.

Tabulka č. 3.1: Zastoupení verzí OS AIX ve zkoumaném prostředí [Vlastní zpracování]

Verze	Počet	Podíl
AIX 5.*	115	3.60%
AIX 5.1	11	0.34%
AIX 5.2	32	1.00%
AIX 5.3	212	6.64%
AIX 6.*	77	2.41%
AIX 6.1	1800	56.37%
AIX 7.*	138	4.32%
AIX 7.1	806	25.24%
AIX 7.2	2	0.06%

[6]U některých serverů nebyla zjištěna přesná verze - značeno '*'.

3.2.4 Ukládání dat

AIX je unixový systém, a tudíž i principy ukládání dat jsou založené na těch z UNIX. Pochopení těchto principů je ale pro správu operačního systému AIX důležité, a proto je nyní rozeberu.

Základem ukládání dat jsou, jako ostatně v každém OS, fyzická uložiště dat (*Physical Volume* – PV). Lze využít pevné disky, SSD, DVD ale také magnetické pásky nebo zařízení dostupná po síti.

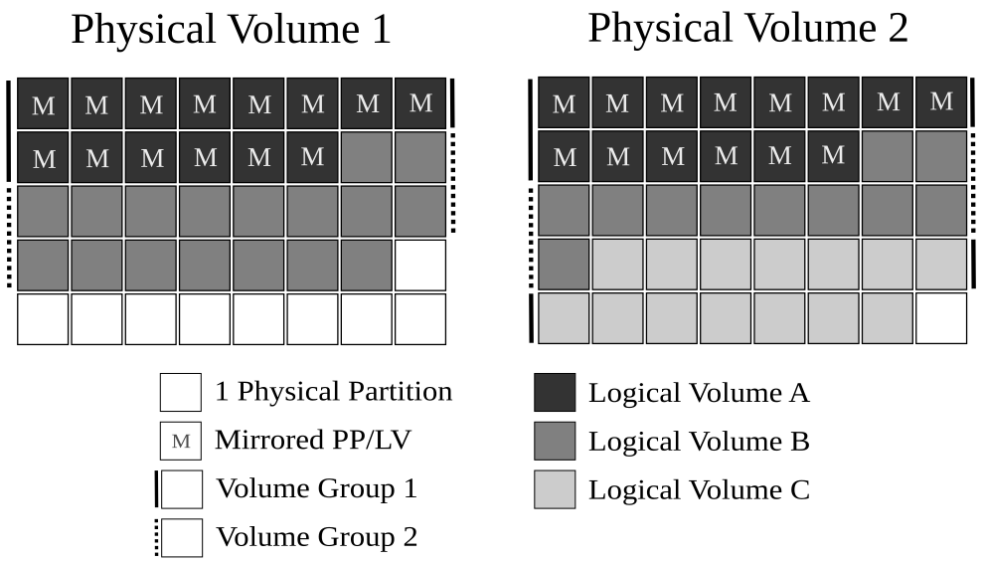
Fyzická zařízení musí být před použitím rozdělena do logických jednotek. Nejmenší z těchto jednotek je fyzický oddíl (*Physical Partition* – PP). Jedná se o nejmenší alokovatelnou jednotku, pro fyzické ukládání dat. PP jako takové nemají standardní velikost. Velikost PP je obvykle 32 nebo 64 MB. Logické oddíly (*Logical Partitions* – LP) jsou nejmenší alokovatelnou jednotkou pro logické ukládání dat. V praxi se rozdíl mezi LP a PP projevuje u zrcadlení. Budeme-li zrcadlit blok dat o velikosti jednoho oddílu, tak tento blok bude zabírat velikost jednoho LP, ale dvou PP. Fyzicky tedy bude použita dvojnásobná část kapacity PV.

Logické svazky (*Logical Volume* – LV) představují shluk LP, ke kterému se přistupuje jako k jednomu celku, a který v danou chvíli používá jeden souborový systém. Všechny LP v LV mají stejnou velikost. Každý adresář (složka) má přiřazený logický svazek, do kterého ukládá data. Pokud je tento svazek plný, tak do něj nelze uložit další data. V případě, že je ale na PV volné místo, tak je možné LV zvětšit.

Skupina svazků (*Volume Group* – VG) je skupinou logických svazků. Logické svazky v jedné skupině svazků se mohou nacházet na různých discích, pokud jsou všechny tyto disky definované v dané VG. Jednotlivé LV mohou mít různé souborové systémy. První (systémem) vytvořenou skupinou svazků je *rootvg*. Obsahuje všechny LV potřebné ke spuštění systému.

Souborové systémy (*File system* – FS) určují, jakým způsobem jsou data zapisována do LV a k čemu daný LV slouží. Zároveň také mohou určit, jak bude daný svazek využíván (pro boot systému, swap...) [3].

Obrázek 3.2 slouží jako příklad, pro lepší pochopení dělení PV do jednotlivých logických jednotek. Náš systém obsahuje dva disky, každý o velikosti 40 PP (každý čtvereček představuje jeden fyzický oddíl – 1 PP). Při velikosti jednoho oddílu 32 MB, by tedy každý disk měl kapacitu 1280 MB. Tento systém obsahuje tři logické svazky, které jsou přidělené do dvou skupin svazků. *VG 1* obsahuje *LV A* a *LV C*. *VG 2* obsahuje *LV B*. *LV A* je zrcadlená na druhý disk. Její velikost je poté 14 LP, což je ale kvůli zrcadlení 28 PP. *LV B* je rozdělena na oba disky, ale zrcadlena není.



Obrázek č. 3.2: Princip ukládání dat [Vlastní zpracování]

Zrcadlení logických svazků

AIX, stejně jako jiné operační systémy, podporuje zrcadlení dat, pro případ poruchy fyzického nosiče dat. Zrcadlení se tedy většinou provádí tak, aby zrcadlené svazky byly na různých PV. Zrcadlit lze jakýkoliv LV nebo VG. Při zrcadlení VG však dochází k zrcadlení celé VG pouze logicky, protože systém prakticky zrcadlí každý svazek samostatně [3].

Typy logických svazků a souborových systémů

Existuje několik druhů logických svazků:

- *boot logical volume* – obsahuje data pro zavedení operačního systému,
- *dump logical volume* (značen jako *sysdump*) – slouží pro uložení stavu kernelu v případě selhání systému,
- *paging logical volume* – swapovací oddíl,
- *raw logical volume* – jsou kontrolované (zpravidla databázovými) aplikacemi,
- *journal logical volume*[7] – ukládá uživatelské soubory do (4KB) bloků,
- *log logical volume* – jsou využívány žurnálovacími svazky pro ukládání logů [3].

Typy LV úzce souvisí s typy souborovými systémy, a dokonce se částečně překrývají. Rozdíly nejsou pro tuto práci důležité, a proto se zde jimi nebudu zabývat. Tabulka 3.2 popisuje souborové systémy, které AIX standardně tvoří.

[7] Žurnálovací logický svazek

Tabulka č. 3.2: Výchozí souborové systémy [3]

Logical Volume	File System	Popis
hd1	/home	obsahuje adresáře uživatelů
hd2	/usr	adresář pro uložení kódu příkazů, a programů
hd3	/tmp	uložiště dočasných souborů
hd4	/	root
hd5		boot logical volume
hd6		paging logical volume
hd8		primární logovací svazek
hd9var	/var	uložiště logovacích souborů a proměnných
hd10opt	/opt	uložiště dalších programů
proc	/proc	FS pro podporu multitaskingu

3.2.5 Dump

Dump je mechanismem operačního systému AIX, který při závažné chybě systému (přesněji řečeno *kernelu*) vytvoří obraz stavu (paměti) jádra ve chvíli selhání, a ten je následně uložen pro případné přezkoumání [26].

Dump Device

Dump Device (DD) je dočasným uložištěm pro dump. Jedná se o samostatný LV, kam AIX po pádu zapíše obraz paměti kernelu.

AIX je schopen práce se dvěma DD - primárním a sekundárním. *Primary dump device* je vytvořen automaticky během instalace, v LV *hd6*. Jedná o *paging logical volume* (viz. tabulka 3.2). *Secondary dump device* slouží jako záložní uložiště pro případ, že by dump nebylo možné uložit do primárního DD. AIX jej během instalace nevytváří, a eviduje jej jako: */dev/sysdumpnull* (tedy jako nenastavené).

Dump Device LV může být různých typů. Většinou se ovšem používá *sysdump*. Dále se používá typ *paging* [26].

Copy Directory

Copy Directory je uložištěm pro dump, před jeho dalším zpracováním. Během instalace je nastavena na */var/adm/ras* [26].

Dump

Po pádu systému a vytvoření obrazu kernelu, se AIX nejprve pokusí obraz umístit do primárního DD. V případě, že to není možné, tak se AIX použije sekundární DD[8]. Následuje restart systému. Třetí krok, kterým je zkopírování dumpu do copy directory, je nutné provést, pokud se dump device nachází na *paging LV*[9]. Administrátor systému může tento krok přeskočit, ale v takovém případě je dump ztracen. Pokud je dump větší než dostupné místo v copy directory, tak administrátor systému dostane možnost dump zapsat na jiné médium nebo pokračovat ve startu systému (a tím dump efektivně zahodit).

Dump systému tedy probíhá ve dvou (potažmo třech) fázích:

1. zápis obrazu dumpu do *Dump Device*,
2. restart systému,
3. zkopírování obrazu dumpu do *Copy Directory* [26].

Doporučení pro administrátory

Obecným doporučením je vytvořit DD o velikosti 1/4 paměti serveru, nebo velikost odhadnout pomocí příkazu *'sysdumpdev -e'*. Rovněž není doporučené zrcadlení DD (pro případ, že OS by selhal právě kvůli zrcadlení). Lepším řešením je ovšem nastavit sekundární DD.

Ve výchozím (tradičním) nastavení se *primary dump device* nachází na *paging LV*. Tento přístup se nedoporučuje ze dvou důvodů: swapovací oddíl nemusí mít během pádu dostatek kapacity pro přijetí dumpu, a protože během restartu je nutné dump zkopírovat do copy directory. Druhý zmíněný problém spolu nese další rizika (například zahození dumpu administrátorem kvůli zkrácení doby výpadku) [26].

3.2.6 Hlášení chyb v AIX

AIX obsahuje mechanismy pro sběr a hlášení zpráv od nejrůznějších částí OS a dalších aplikací. Ve výchozím stavu jsou v systému nainstalované 2 řešení: *syslog* a *errdemon*.

Syslog je standardní řešení pro sběr všech hlášení, které se v systému mohou vyskytnout. Základen je *syslog deamon*, který zaznamenává všechny zprávy vytvořené přes funkci *syslog()*. Všechny přijaté zprávy následně třídí do příslušných logovacích souborů.

Errdemon je démon pro sběr chybových zpráv (ze souboru */dev/error*), a jejich zápis do systémového logu. Tento démon může dále vytvářet upozornění na chyby.

[8] Pokud není možné dump uložit vůbec, tak jej systém zahodí.
[9] OS tento oddíl používá pro swapování.

Každá zpráva systému AIX obsahuje několik standardních údajů, jako je datum a čas zápisu, typ a popis problému nebo také identifikátor chyby (*error ID*). K výpisu posledních chyb poté slouží příkaz *errpt*, který vypíše zaznamenané chyby se všemi příslušnými údaji [3], [10].

3.2.7 The largest dump device is too small

'The largest dump device is too small'[10] je jednou z méně častých chyb, které v systému AIX mohou nastat. Popis chyby přesně vypovídá o daném problému: v případě pádu systému, by došlo ke ztrátě dumpu. Důvodem by bylo nedostatečné místo na obou DD. Nejčastějším řešením je rozšíření LV, které je nastaveno jako DD. Identifikátory této chyby jsou:

- číslo chyby – *E87EF1BE*,
- popis chyby – *'The largest dump device is too small'* [27].

Tato chyba byla ve zkoumané prostředí doposud řešena manuálně - pracovníky service desků. V následující kapitole proto popíši tvorbu programu pro automatické řešení tohoto problému.

Četnost výskytu

Chyba *E87EF1BE* se řadí mezi méně časté chyby. To jsem ověřil ve sledovaném prostředí. Za dva měsíce bylo na 1000 sledovaných serverech zaznamenáno 91 případů této chyby. Výpočtem podle následujícího vzorce jsem došel k závěru, že zkoumaná chyba se na serveru vyskytuje jednou za 1,81 let (tedy jednou za necelých 22 měsíců).

$$Stredni\ doba\ mezi\ vyskyty = \frac{dnu\ v\ obdobi}{365} * \frac{serveru\ celkem}{pocet\ vyskytu} \tag{3.1}$$

Dumpcheck

Dumpcheck je příkazem, který kontroluje velikost DD, a může zalogovat chybu. Ve výchozím nastavení se automaticky provede každý den v 15 hodin. Prakticky se jedná o příkaz, který loguje chybu 'The largest dump device is too small'[11] [10].

[10] Největší dump device je příliš malý.

[11] Řešení chyby 'The largest dump device is too small' můžeme označit jako reaktivní i proaktivní zásah. Administrátor sice reaguje na chybu systému (ticket), ale ten vznikl v důsledku proaktivního skenu systému.

3.2.8 Správa systému

Správa systémů AIX je možná několika způsoby, přes terminál, terminálové rozhraní *smitty* nebo grafické uživatelské rozhraní.

Grafická uživatelská rozhraní

AIX podporuje několik unixových grafických uživatelských rozhraní (GUI[12]). Výchozím rozhraním je Common Desktop Environment (CDE). AIX dále umožňuje instalaci (XDM) X-windows Display Manager nebo linuxových rozhraní jako KDE a Gnome přes *AIX Toolbox for Linux* [23], [28].

SMITTY

SMIT (System Management Interface Tool) je interaktivní nástroj pro správu AIX, který je dostupný v plně terminálové i grafické verzi. Administrátor může přes SMIT téměř plně ovládat systém a to bez znalosti příslušných příkazů, potřebuje k tomu ovšem příslušná práva. Tyto příkazy poté AIX provede za administrátora, a může je rovněž vypsat. Terminálovou verzí SMIT je SMITTY. Spouští se přes příkaz *smitty*.

```
                  Change / Show Characteristics of a User

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                 [Entry Fields]
* User NAME                                          guest
  User ID                                           [100]                #
  ADMINISTRATIVE USER?                               false               +
  Primary GROUP                                     [usr]                +
  Group SET                                         [usr]                +
  ADMINISTRATIVE GROUPS                             []                   +
  ROLES                                             []                   +
  Another user can SU TO USER?                       true                +
  SU GROUPS                                         [ALL]                +
  HOME directory                                    [/home/guest]
  Initial PROGRAM                                   []
  User INFORMATION                                  []
  EXPIRATION date (MMDDhhmmyy)                      [0]
[MORE...52]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Obrázek č. 3.3: SMITTY [29]

[12]Graphical User Interface

Shell

AIX rovněž umožňuje ovládání systému přes terminálové příkazy (shell). Výchozím shellem v tomto systému je *ksh*, a to verze *ksh88*. Dostupný je rovněž *ksh93*[13].

Ksh v systému AIX nahradil *bsh* (Bourne Shell), a je s ním zpětně kompatibilní. Dále je v systému dostupný *rksh* (restricted Korn shell).

Pro běžnou administraci systému není rozdíl mezi ksh88 a ksh93 příliš důležitý. Projeví se totiž během psaní skriptů. Ksh93 totiž obsahuje další vestavěné příkazy (například pro práci s poli), a nemá plnou zpětnou kompatibilitu [10].

3.2.9 Vybrané příkazy

sudo

Příkaz *sudo* nepatří na AIX mezi předinstalované balíčky, před použitím je jej tedy nutné doinstalovat [30]. Jednou z odlišností oproti ostatním systémům (Linux...) je, že sudo lze na AIX nutné použít pouze pro privilegované příkazy. Použití sudo pro neprivilegované příkazy, nebo pod uživatelem *root* vede k chybě.

```
1:root@aix53:/root # sudo ls
Sorry, user root is not allowed to execute '/usr/bin/ls'as root on aix53.com
```

Obrázek č. 3.4: Selhání příkazu při nesprávném užití příkazu sudo [Vlastní zpracování]

uname a oslevel

Příkazy *uname* a *oslevel* se používají k ověření běžícího operačního systému. *Uname* lze použít na všech unixových systémech pro zobrazení jména OS (zde *'AIX'*). *Oslevel* se nachází pouze na AIX. Lze jej použít pro zobrazení verze systému [10].

Logical Volume Management

Pro správu LV na AIX existuje několik příkazů, z nichž nyní některé stručně popíši. Většina následujících příkazů je specifická pro AIX.

Příkaz *lslv* slouží k zobrazení údajů o logickém svazku. Umožňuje zobrazit údaje jako je velikost a počet PP, typ LV nebo příslušnost k VG. Jeho obdobou pro volume group je příkaz *lsvg*.

[13] Ksh je zkratkou pro Korn shell, přičemž ksh93 se někdy označuje jako *Enhanced Korn shell*.

Správa *paging LV* se oproti většině ostatních LV liší použitými příkazy. Pro rozšíření 'běžného LV' můžeme použít příkaz *extendlv*. Pro swapovací svazek je vhodné použít příkazů *chps* a *chlv*. *Chlv* se používá pro změnu charakteristik (libovolného) LV. Parametr *-u* nastavuje horní hranici disků pro rozšiřování. Poté je možné bezpečně použít příkaz *chps* pro přidání logických oddílů [10].

Sysdumpdev

Pro výpis údajů o nastavení DD existuje příkaz *sysdumpdev*. Pro tuto práci jsou důležité jeho dva parametry:

- *-l* – vypíše seznam základních údajů o nastavení dump,
- *-e* – podle současné zátěže systému odhadne velikost dumpu (v bytech)[14] [10].

errpt

Příkaz *errpt* umožňuje výpis zaznamenaných chyb. Jeho parametr *-a* vypíše rozšířený popis chyb. Parametr *-j* můžeme použít k výběru specifických chyb přes *errorID* [15] [10].

3.3 Nástroj pro automatizaci správy ICT – IPC

IPC je nástrojem pro automatizaci správy ICT prostředí, který umožňuje správu všech platforem, na které se lze připojit pomocí SSH nebo PowerShell. Ve zkoumaném prostředí je v současné době využíván pro provádění reaktivních a proaktivních opatření.

3.3.1 Vlastnosti nástroje IPC

Nástroj IPC je *bezagentní centralizované řešení*. Na koncových stanicích tedy není potřeba žádný dodatečný SW (agent), a správa je kompletně řízena z nástroje, popřípadě z jeho uživatelského prostředí. K připojení na servery IPC používá *SSH* nebo *PowerShell*, což jsou protokoly implementované na prakticky všech unixových a windowsových operačních systémech.

K připojení jsou třeba údaje o koncové stanici a přihlašovací údaje. Data o koncové stanici jsou uložena v tzv. *CMDB*[16] Uživatelské údaje jsou uloženy v oddělené (šifrované) databázi - *Locksmith*.

[14] Nadstavbou pro příkaz 'sysdumpdev -e' je příkaz *dumpcheck*. Dumpcheck je nutné volat přes jeho plnou cestu: *'/usr/lib/ras/dumpcheck'*.

[15] Například: *E87EF1BE*.

[16] Configuration management database – Databáze pro management konfigurace.

IPC umožňuje reaktivní i proaktivní zásahy. Reaktivní zásahy umožňuje napojení na monitorovací systémy – tzv. *Radar*. Radar obdrží všechny tickety předtím, než jsou odeslané na service desk. V případě, že na daný ticket existuje (a je povolen) automatizační program – tzv. *automat* – tak je daný program automaticky spuštěn. Proaktivní zásahy jsou spouštěny manuálně, nebo v předem nastaveném čase. Automaty jsou prováděny pomocí *Execution Engine*, který se stará o jejich správný běh.

Nástroj IPC je možné připojit na více zákaznických infrastruktur (a tedy i Monitorovacích infrastruktur) současně.

Bezpečnost nástroje IPC

Technické zabezpečení nástroje IPC je zajištěno na několika úrovních. Nástroj je umístěn uvnitř chráněné firemní sítě, a tudíž není dostupný z internetu. Do samotného nástroje mají přístup pouze oprávnění uživatelé, jejichž práva jsou přidělována (a revokována) dle jejich pole působnosti. Data i samotné instance nástroje jsou zálohovány. Veškeré procesy týkající se používání nástroje podléhají příslušným firemním procesům, na které jsou zaměstnanci školeni.

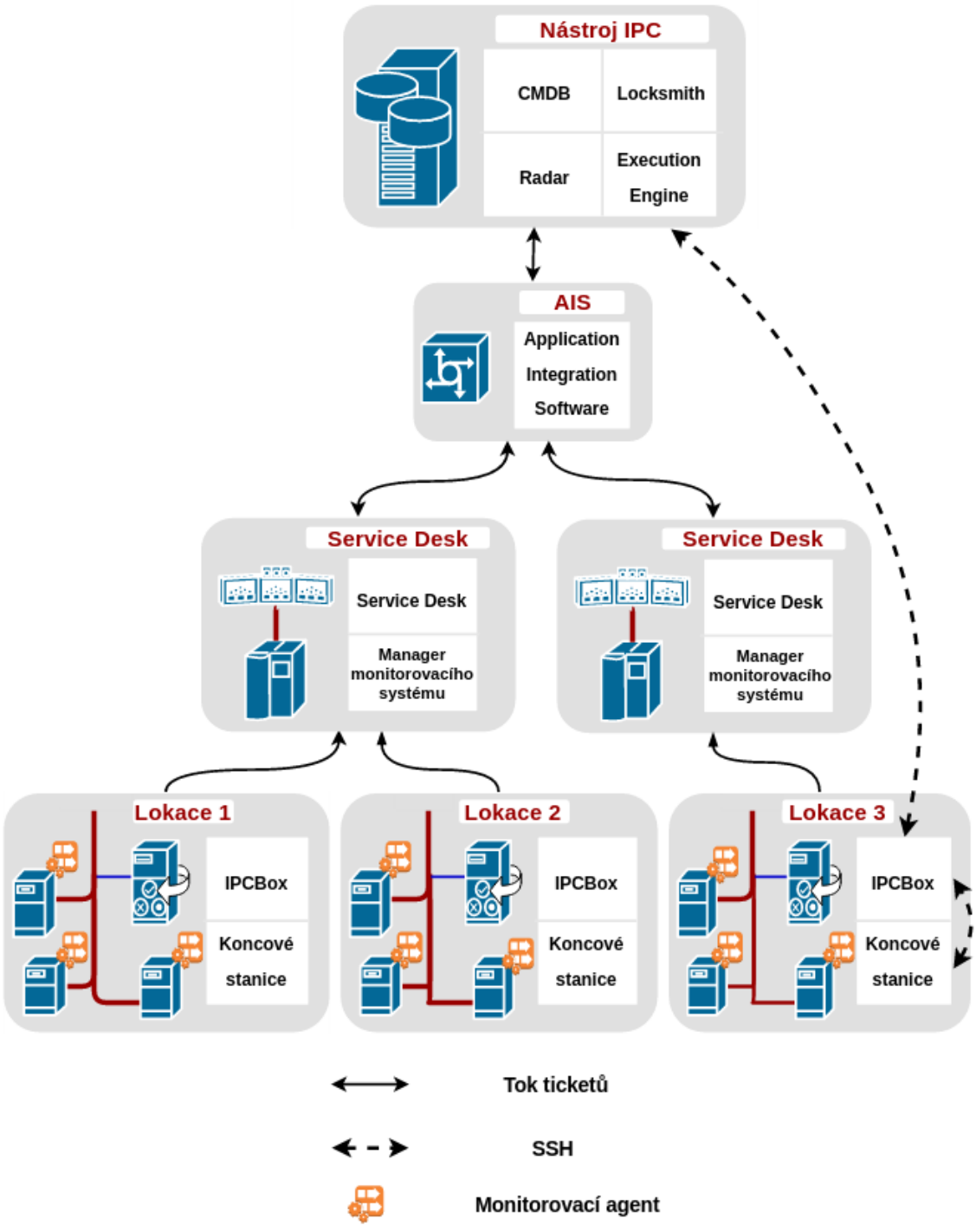
Nástroj se do zákaznických sítí (které jsou rovněž chráněné) připojuje pomocí VPN. Připojení na spravované servery je poté zabezpečeno pomocí protokolů SSH a PowerShell. Veškeré přístupové údaje pro připojení na servery jsou uložené v databázi *Locksmith*. Do té mají přímý přístup pouze vybraní zaměstnanci.

3.3.2 Infrastruktura nástroje IPC

IPC lze napojit na monitorovací infrastrukturu, která by měla být standardizovaná. Toho lze dosáhnout přímým napojením, nebo pomocí AIS[17]. AIS v tomto případě obstarává překlad ticketů z více monitorovacích řešení do jednotného formátu (srozumitelného pro Radar). Pro připojení na koncové zařízení se používá tzv. *IPCBox*. Jedná se o server, který pro IPC slouží jako vstupní brána do prostředí v dané lokaci (datacentru).

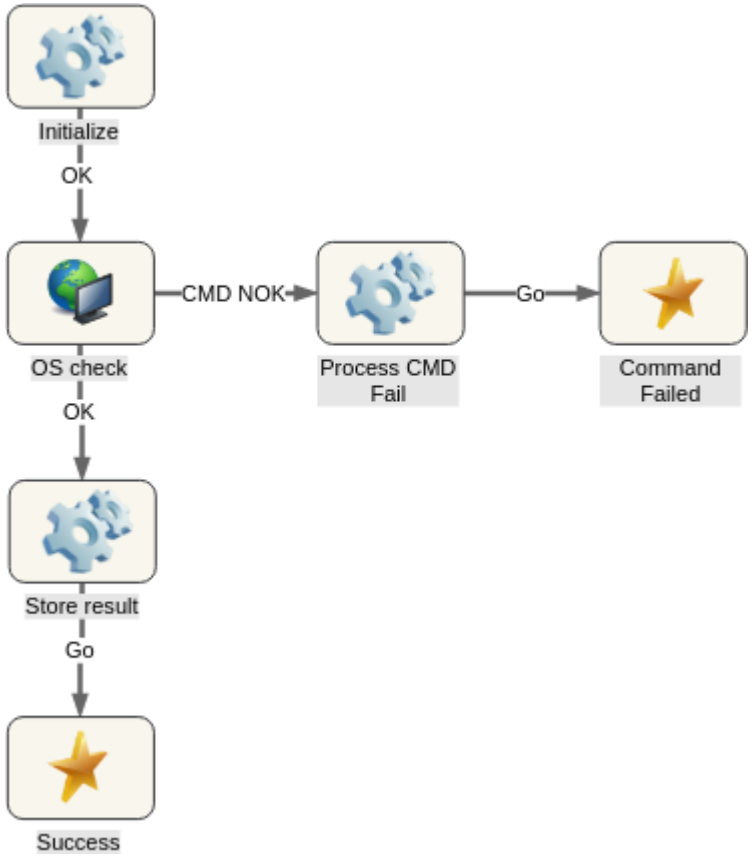
Propojení nástrojů popisuje obr. 3.5. Celá reakce na událost začíná ve chvíli, kdy server vytvoří ticket, který odešle do monitorovacího nástroje. Monitorovací nástroj poté předá ticket do AIS, který jej pošle do nástroje IPC. IPC po obdržení ticketu ověří, jestli je pro daný ticket dostupný automat. V případě, že je, tak je daný automat spuštěn (jinak je ticket odeslán zpět do monitorovacího nástroje). Automat se připojí přes daný protokol (SSH nebo PowerShell) na IPCBox, který mu předá spojení na server. Automat poté provede naprogramovanou činnost, a spojení ukončí. Pokud se problém podařilo vyřešit,

[17] Application Integration Software – SW pro integraci aplikací.



Obrázek č. 3.5: Infrastruktura nástroje IPC [Vlastní zpracování]

tak je ticket uzavřen. V obou případech dojde k aktualizaci ticketu a odeslání zpět do monitorovacího nástroje, kde dochází k dalšímu zpracování.



Obrázek č. 3.6: Jednoduchý automat pro kontrolu OS [Vlastní zpracování]

3.3.3 Automat

Nástroj IPC je postaven v jazyce Java. K návrhu automatů se ale využívá grafického vývojového prostředí, kde se automat tvoří z předpřipravených bloků - *stavů* a *přechodů*. Samotný automat tedy funguje na principu konečného automatu. V některých stavech je možné definovat či upřesnit další funkcionalitu pomocí příkazů, spustitelných na serveru, nebo pomocí JavaScriptu. Součástí každého automatu je rovněž *Matcher*, proměnné a *spojení*.

Proměnné

Každý automat umožňuje definici proměnných pro ukládání stavu automatu a řízení jeho toku. Každá proměnná musí mít výchozí hodnotu, přičemž je ji možné využít ve většině bloků.

Vybrané stavy

Automat je možné skládat z asi 30 různých stavů, přičemž zde zmíním pouze některé:

- *CMDB loopkup* – vyhledá informace v CMDB,
- *Ticket update* – přidá informace do ticketu, který spustil automat,
- *Send email* – slouží k odeslání emailu,
- *Edit Variable* – umožňuje úpravu hodnoty proměnné pomocí JavaScriptu, nebo matematických operací,
- *Script* – provádí vývojářem definovaný JavaScript kód,
- *Host Command* – provede příkaz nebo skript na koncovém zařízení. Skript nebo příkaz musí být serverem podporované,
- *Link* – umožňuje vnoření automatů s předáním spojení a proměnných,
- *Noop* – nemá žádnou funkcionalitu a slouží ke zvětšení přehlednosti,
- *Success* – ukončuje automat, musí být posledním spuštěným stavem.

Přechod mezi stavy

Tok automatu může vývojář ovlivnit podobně, jako ve vývojovém diagramu – přechody (šipkami). Ze všech stavů (kromě Success) může vycházet více přechodů, automat ale vždy vybere jeden, a to podle definované podmínky. Přechody mohou být několika základních typů:

- *Always* – je výchozím přechodem, který je platný vždy,
- *Variable* – je platný, pokud nastavená proměnná má zadanou hodnotu. Rovněž umožňuje kontrolu výstupu předchozího stavu, pomocí zvláštní proměnné *StdRet* (0 značí úspěch),
- *Boolean Expression* – umožnuje tvorbu složitější podmínky pomocí výrazů booleovy algebry,
- *JavaScript* – je pro složitější vyhodnocení (např.: drobná úprava textu).

Platnost přechodů lze rovněž ovlivnit pomocí operátorů *negate* (negace hodnoty) a *weight* (váhy přechodu). Váhu přechodu je vhodné použít, pokud by přechody mohly být platné ve stejnou chvíli (v takové situaci dojde v výběru přechodu s vyšší hodnotou).

Connections

Každý automat se musí připojit na server, k čemuž slouží *Connections* (Spojení). Spojení jsou definována centrálně, a poté přiřazeny k automatům. Každé spojení definuje vybraný IPCBox, a v automatu poté dojde k dynamickému přiřazení cílového serveru.

Matcher

Tzv. *Matcher* je mechanismem pro výběr automatu. Jedná se o regulární výraz, který je použit pro přiřazení automatu k ticketu. Matcher je považován za platný, pokud jeho regulární výraz spuštěný proti textu ticketu vrátí pravdivou hodnotu.

Matcherů lze pro jeden automat použít více (a spojit je pomocí logických operací AND a OR), popřípadě je negovat. Rovněž je možné definovat část ticketu, proti které se bude matcher ověřovat.

Po přijetí ticketu nástrojem IPC jsou matchery všech automatů ověřeny proti ticketu. Spuštěny jsou pouze automaty, které jsou schválené pro produkční nasazení, a podmínka vytvořená matchery je splněna.

Matcher je tvořen pro každého zákazníka zvlášť na základě nastavení konkrétního monitorovacího systému. Proto jeho tvorbu v této práci popisovat nebudu.

Logické rozdělení automatu

Vyvíjené automaty je rovněž vhodné logicky rozdělit, na specifickou část (tzv. *wrapper*) a část obecnou (tzv. *komponenta*).

Wrapper je specifický pro monitorovací infrastrukturu. Je připraven ze vzoru, a následně upraven podle potřeby. Obsahuje nastavení, funkcionalitu potřebnou k připojení se na server, matcher, a práci s ticketem. Rovněž spouští generickou část (přes blok *Link*).

Komponenta (někdy také značená jako automat) je, jak již bylo řečeno, generickou částí automatu. Určuje postup zásahu na serveru a vyhodnocení, jestli byl problém vyřešen (a může tedy dojít k uzavření ticketu). I komponenta ale může obsahovat další vnořené komponenty.

Automat, který budu popisovat v další kapitole je dle tohoto dělení komponentou. Pro zjednodušení budu ale používat pouze označení *automat* a vnořené komponenty pouze jako komponenty.

3.4 Požadavky na navrhované řešení

Před samotným návrhem řešení, jsem musel zajistit, aby výslednou automatizaci (pro řešení chyby 'The largest dump device is too small') bylo možné nasadit pro co největší část zákazníků společnosti. Proto jsem musel vytvořit seznam požadavků, které výsledný automat musí splňovat. Ty jsem stanovil na základě oficiální dokumentace a doporučení

pro správu AIX (což shrnuje kapitola o OS AIX). Dále jsem vycházel z nároků kladených na řešení zákaznickými service desky. Po přidání požadavků pro tvorbu automatů a konzultaci s odborníky na správu AIX, jsem získal následující požadavky:

- Automat musí zvětšit oba DD, pokud to situace vyžaduje.
- Podpora pro verze systému s rozšířenou podporou.
- Každý zákazník může mít jinak nastavené podmínky pro správu, a tudíž musí být dostupná možnost parametrizace nastavení.
- Prvky v každé infrastruktuře mohou mít předepsané nastavení a automat musí být schopen detekovat nastavení rozdílné.
- Režim, kdy automat neprovádí žádné zásahy, a pouze poskytuje přehled o aktuální situaci.
- Automat musí podporovat příkaz *sudo*.

3.5 Souhrn analýzy

Tuto práci tvořím pro firmu IBM, která poskytuje IT služby svým zákazníkům. Jednou z těchto služeb je správa ICT infrastruktury, a proto jsem tuto kapitolu začal stručným popisem prostředí, pro který je program navržen.

Stěžejní částí této kapitoly je popis operačního systému AIX (unixový systém od firmy IBM). Důraz jsem věnoval zejména systému ukládání dat, mechanismu ukládání stavu paměti počítače po selhání kernelu (dump), hlášení chyb a správě tohoto OS, včetně několika vybraných příkazů. Témata této sekce, byla vybrána s ohledem na řešení chyby *'The largest dump device is too small'*, což je specifický problém správy OS AIX. Tuto chybu AIX hlásí, pokud příkaz *dumpcheck* zjistí nedostatečné místo v jednom z Logických svazků určených pro ukládání dat dumpu. Řešením právě tohoto problému se budu zabývat v následující kapitole.

Předposlední část této kapitoly byla věnována popisu nástroje IPC, který firma IBM využívá k autonomní správě IT prostředí, a který proto použiji k automatickému řešení již zmíněné chyby. Popsal jsem základy fungování tohoto nástroje, jeho začlenění do stávajících ICT infrastruktury a důležité prvky vývoje automatizačních programů – automatů. Kapitolu jsem poté zakončil krátkým souhrnem požadavků na tento automat.

4 VLASTNÍ NÁVRH ŘEŠENÍ, PŘÍNOS PRÁCE

V předchozí kapitole jsem se zabýval analýzou problému, který nyní budu řešit. Je jím návrh automatu pro automatické řešení problému s nedostatečně velkým dumpovacím prostorem na serverech s operačním systémem AIX. V dalších částech této kapitoly provedu ekonomické zhodnocení implementovaného řešení, a v závěru kapitoly popíši další prvky automatické správy ICT prostředí.

4.1 Automat Dump Device Extender

Navrhovaný automat se jmenuje *Dump Device Extender*[1]. Jeho cílem je reakce na ticket *'The largest dump device is too small'*, a tedy zvětšení *dump device* logických svazků, a pokud je to možné tak uzavřít ticket.

Automat tedy provede 3 hlavní kroky – diagnostiku daného serveru, vyhodnocení situace, a rozšíření *dump device*, pokud to situace umožňuje. V každém případě automat do ticketu přidá popis stavu serveru po ukončení jeho činnosti.

Automat jsem rozdělil do 4 programových celků (automatů):

- *Dump Device Extender* – hlavní část automatu. Obsahuje hlavní logiku programu a část diagnostických příkazů, a spouští obě vnořené komponenty.
- *Scan Dump Device* – zjišťuje stav jednoho DD.
- *Work on Dump Device* – provádí rozšíření konkrétního DD.
- *Wrapper* – generická část automatu, kterou se v této práci nebudu zabývat. Pro navrhovaný automat zajistí spojení se serverem, aktualizaci ticketu, a předá nastavení automatu pro daného zákazníka, které předá ve formátu JSON.

Návrh automatu v nástroji IPC

Tento automat je navrhován a implementován v nástroji IPC, ale tuto proceduru je možné implementovat i v dalších nástrojích. Před popisem funkcionality automatu zmíním několik důležitých poznámek.

Automat používá pouze několik druhů stavů (state) nástroje IPC[2]. Konkrétně se jedná o stavy *Script*, *Host Command*, *Link* a *Success*. Veškerý zmíněný JS kód a většina rozhodovací logiky je prováděna právě uvnitř stavů se skripty. Příprava příkazů (viz. kapitola 4.1.1 je rovněž prováděna uvnitř těchto stavů, přičemž příkaz je vždy uložen do proměnné.

[1] U názvů automatů a proměnných ponechám původní anglické názvy. Komentáře kódu jsou přeloženy.
[2] Popis těchto principů je v kapitole 3.3.3.

Tato proměnná je poté použita v následujícím *Host Command* stavu, který její hodnotu vyhodnotí jako příkaz a provede jej[3].

Volání komponent *Scan Dump Device* a *Work on Dump Device* je provedeno pomocí stavů *Link* (volání automatu z wrapperu probíhá stejným způsobem). Linky také předávají všechny potřebné proměnné a připojení k serveru. Každá větev automatu musí končit stavem *Success*.

V automatu používám pouze přechody *Always* a *Variable*. Druhý zmíněný stav používám pro větvení na nejvyšší úrovni automatu, např. pro opakované spouštění komponent nebo pro ověření správného provedení příkazu. Pokud nemůže nastat jiná situace, tak je použit přechod *Always*.

4.1.1 Parametry automatu

Automat přijímá parametry ve formátu JSON zapsané do jediné proměnné. Výhodou JSONu oproti běžným proměnným je, že jeho obsah lze měnit podle aktuální potřeby (a tedy uložit libovolný počet parametrů). Výchozí stav parametrů zachycuje výpis kódu 4.1. První parametr je zde uveden pro zapnutí podpory systémů AIX, které již nejsou plně podporovány (AIX 5.1, 5.2 a 5.3). Parametry *s_diagnosis_mode* a *s_diag_close_fake* jsou zde pro zákazníky, kteří nedovolují automatické zásahy na serverech. První přepíná automat do diagnostického módu (kdy bude provedena pouze analýza). Další parametr poté umožňuje uzavřít ticket, pokud je zapnutý diagnostický mód, ale zásah na serveru není potřeba.

```
{
  "s_remediate_older_AIX": false,
  "s_diagnosis_mode": false,
  "s_diag_close_fake": true,
  "s_extend_only_largest": false,
  "s_max_DD_size_threshold": 100,
  "s_continue_on_extend_fail": true,
  "s_remediate_small_cpFldr": true,
  "s_requirements": []
}
```

Výpis č. 4.1: Parametry programu ve formátu JSON [Vlastní zpracování]

Dalším parametrem je přepínač *s_extend_only_largest*. Ten umožní rozšiřování obou LV, pokud nejsou dostatečně velké. V opačném případě dojde k rozšíření pouze větší LV.

[3] Všechny uvedené programové výpisy jsou zkrácené. Jsou vynechané řádky, které pro tuto práci nejsou důležité (v případě výpisů příkazů), nebo které se opakují.

Následující parametr nastavuje maximální část (v procentech), kterou může DD zabrat na jeho VG. Klíč *s_continue_on_extend_fail* umožňuje automatu provést rozšíření druhého DD, pokud by rozšíření prvního LV selhalo. Další parametr umožňuje zavření ticketu v případě, kdy je *copy directory* příliš malá pro uložení dalšího výpisu.

Posledním parametrem, který automat běžně přijímá jsou požadavky zákazníka. Pokud server nesplňuje tyto požadavky, tak bude tato skutečnost přidána do obsahu ticketu, a automat provede pouze diagnózu. Tento parametr je pole JSONů, jehož každý prvek má strukturu, kterou ukazuje výpis 4.2. Požadavky jsou splněny, pokud server splňuje plně jeden řádek, anebo parametr není vůbec vyplněn.

```
{
  "tier": "n/a",
  "mAddressMask": "x.x.x.x",
  "lvType": "n/a",
  "mirroredLV": "n/a",
  "ddsSet": "n/a",
  "sameSize": "n/a",
  "ddsOnSameVG": "n/a"
}
```

Výpis č. 4.2: Požadavky zákazníka ve formátu JSON [Vlastní zpracování]

První dva parametry určují tzv. 'tier' serveru (produkční úroveň[4]) a rozmezí IP adres monitorovaného serveru. Tímto způsobem lze stanovit servery, kterých se pravidlo týká. Další pravidla poté stanovují požadovaný typ LV, povolení zrcadlení, počet nastavených *dump device*. Předposlední pravidlo (*sameSize*) stanovuje, jestli obě zařízení mohou mít stejnou velikost (tedy, že vždy dochází k zápisu na oba LV). Poslední pravidlo stanovuje, jestli mohou být DD na stejné VG.

Funkce pro přípravu příkazů

Jedním z požadavků na automat je možnost používání příkazu sudo. Funkce, kterou jsem kvůli tomu vytvořil je na výpisu 4.3. Umožňuje použití nejen standardního nastavení, ale také zákaznické nastavení, které lze přidat do JSONu s parametry, jako klíč *s_sudoCommands*. Tuto funkci automat volá vždy před příkazovými bloky, pro přípravu potřebných příkazů[5].

[4] Například produkce, nebo testovací servery.

[5] Tato funkce nerozhoduje, jestli je automat používá sudo nebo běží pod uživatelem root. To spadá do funkcionality wrapperu, který podle nastavení nastaví proměnnou *sudo_cmd*.

Funkce potřebuje 3 parametry: příkaz, který má být proveden, jméno příkazu a jméno proměnné, ve které bude příkaz uložen pro vložení do bloku *Host Command*. Pokud funkce nalezne v parametrech klíč *s_sudoCommands*, tak bude pracovat s nastavením sudo od zákazníka (zároveň bude provedena kontrola, jestli byl obsah převeden na objekt, nebo se stále jedná o řetězec – JSON). Pro opačný případ, je ve funkci nadefinován JSON s výchozím nastavením. Následně se na základě hodnoty klíče s nastavením do příkazu přidá příkaz sudo nebo ne. Poslední akcí je přiřazení příkazu (jeho textové podoby) do vybrané proměnné.

```
addSudo = function (cmd, paramName, cmdVarName) {
    var sudo = '';
    if (typeof s_sudoCommands !== 'undefined') { // použij
   nastavení zákazníka
        if (typeof s_sudoCommands === 'string') { // konverze
   na objekt:
            s_sudoCommands = JSON.parse(s_sudoCommands);
        }
        var commands = s_sudoCommands.get(paramName)
    } else { // použij standardní nastavení:
        var commands = { sysdumpdev: true, lslv: false,
            lsvg: false, errpt: false, extendlv: true,
            chlv: true, chps: true, df: false };
    }
    if (commands[paramName] === true) {
        sudo = sudo_cmd;  // přidej sudo
    }
    setVariable(cmdVarName, sudo + ' ' + cmd); //nastav příkaz
   do proměnné
};
```

Výpis č. 4.3: Skript pro přidání příkazu sudo [Vlastní zpracování]

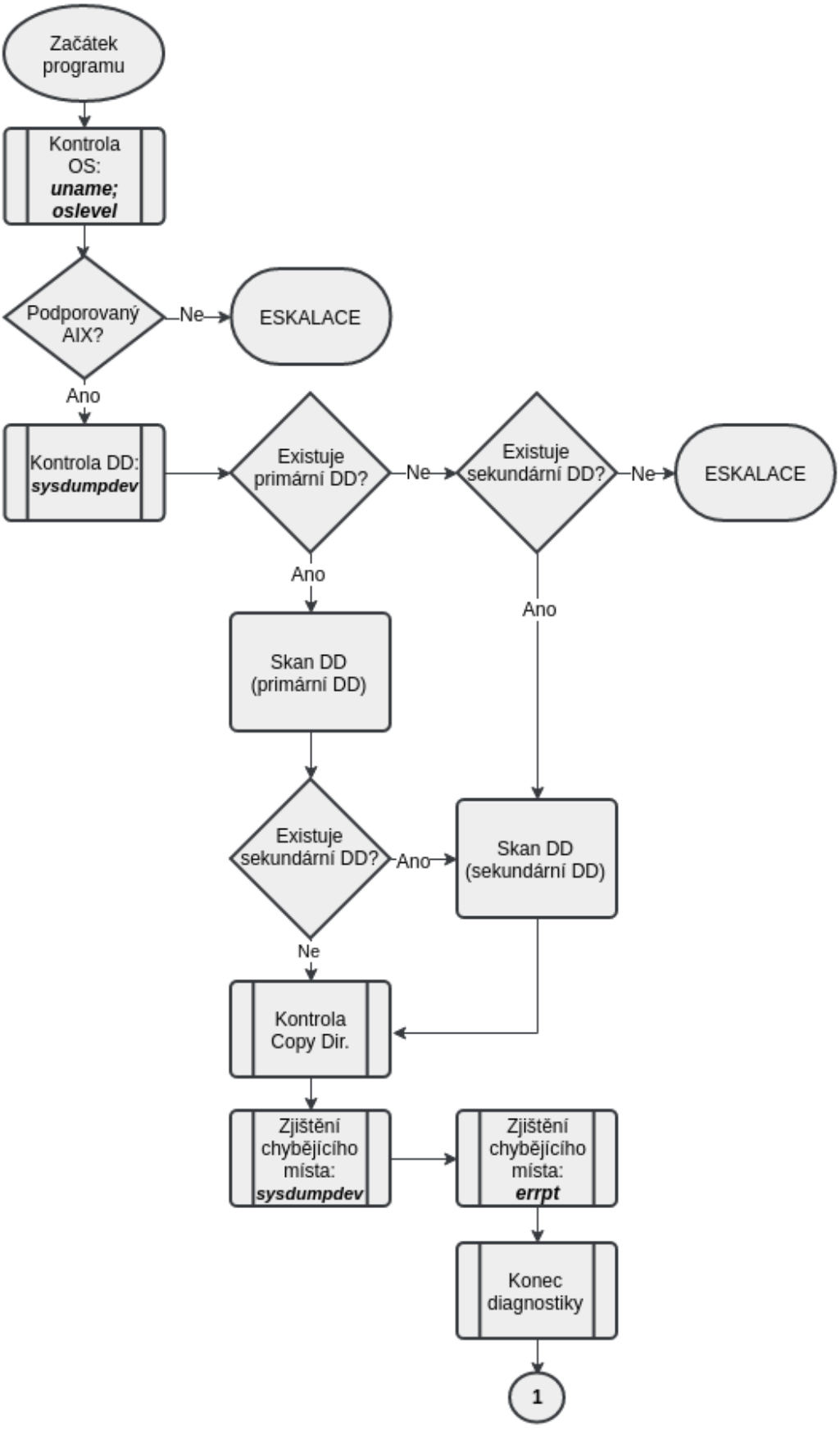
4.1.2 Hlavní tok programu – úvodní kontrola

Nyní se dostáváme k první části samotného automatu. Tu zobrazuje vývojový diagram na obrázku 4.1.

Prvním krokem, který automat provádí je ověření OS. Její výsledek je použit pro ukončení programu v případě, že by OS nebyl podporovaný.

```
echo ‘uname‘ ‘oslevel‘
```

Výpis č. 4.4: Příkaz pro ověření verze OS [Vlastní zpracování]



Obrázek č. 4.1: Fungování Dump Device Extender – 1. část [Vlastní zpracování]

Výstup příkazu poté může vypadat například takto: *'AIX 7.1.0.0'*. Pokud se nejedná o podporovanou verzi systému AIX nebo by se o AIX nejednalo vůbec, tak dojde k ukončení programu. Do ticketu by byla přidána potřebná hláška (o nepodporovaném OS).

Dalším příkazem je již úvodní kontrola nastavení *dump device*. Následující výpis ukazuje také ukázku výstupu příkazu.

```
$  sysdumpdev -l
primary              /dev/pdumplv
secondary            /dev/sysdumpnull
copy directory       /var/adm/ras
forced copy flag     TRUE
always allow dump    TRUE
```

Výpis č. 4.5: Zjištění nastavení dump device včetně výstupu [Vlastní zpracování]

Z výpisu vyplývá, že první dva řádky odkazují na logické svazky, na kterých nachází primární a sekundární DD. Další řádek obsahuje cestu ke *copy directory*. Po tomto příkazu dochází k volání skriptu, který výsledek příkazu uloží pro další použití. Poté dochází k určení, která zařízení jsou používána. Ukázka toho, jak tento skript provádí výběr informací z textu je na výpisu 4.6. Ten obsahuje dva regulární výrazy. První (*cutter*) slouží k rozdělení řádku na název položky a její hodnotu, a to podle většího počtu netisknutelných znaků, než je jedna. Obě hodnoty (název položky i její hodnota) jsou vráceny. Pole, které obě hodnoty obsahuje je dále rozděleno pomocí RegExp *getLV*, který vrátí pouze název dané LV (a to podle dopředných lomítek).

Skript dále pokračuje určením, které LV budou kontrolovány. V případě, že by žádný LV nebyl nastaven, tak opět dochází k eskalaci ticketu k operátorovi. Jinak dochází k volání komponenty *Scan Dump Device*, pro první DD.

```
var outputLines = (sysdumpdev_l_Output + '').split("\n");
var cutter = /(.*?)\s{2,}(.+)/; //regEx na získání hodnoty
var getLV = /\/\w+\/(.+)/; //regEx na získání jména LV
primaryDD = (outputLines[0].match(cutter)[2]).match(getLV)[1];
secondaryDD = (outputLines[1].match(cutter)[2]).match(getLV)[1];
copyDirectory = outputLines[2].match(cutter)[2];
setVariable('primaryDD', primaryDD); // ulož do proměnné
```

Výpis č. 4.6: Parsování nastavení dump device [Vlastní zpracování]

4.1.3 Komponenta Scan Dump Device

Úkolem této komponenty (která je zobrazená na diagramu 4.2) je zjištění stavu konkrétního *dump device*. K tomu jsou potřeba dvou příkazů – příkazu *lslv* a *lsvg*. Výstupem je

soupis důležitých hodnot, které jsou budou potřeba v dalších částech automatu.

```
$ lslv pdumplv
LOGICAL VOLUME:   pdumplv     VOLUME GROUP:   rootvg
...
TYPE:             sysdump     WRITE VERIFY:   off
MAX LPs:          512         PP SIZE:        128 megabyte(s)
COPIES:           1           SCHED POLICY:   parallel
LPs:              1           PPs:            1
...
```

Výpis č. 4.7: Zkrácený výstup příkazu lslv [Vlastní zpracování]

Z výpisu 4.7 je jasné, že jej bude třeba opět parsovat. Protože ale kód pro parsování je velmi podobný kódu z výpisu 4.6, tak uvedu pouze data, která musí být z výstupu skriptu získána:

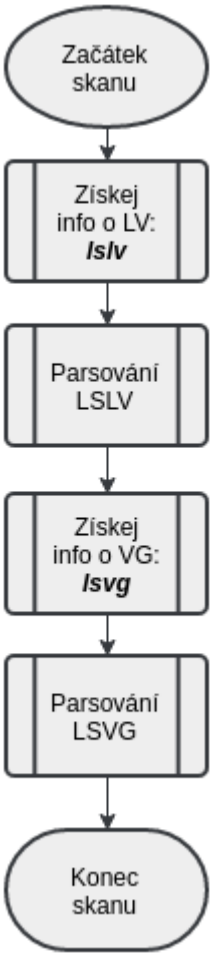
- *VOLUME GROUP* – název VG, která obsahuje dané zařízení,
- *TYPE* – typ logického svazku (souborový systém),
- *LPs* – počet použitých logických oddílů,
- *PPs* – počet použitých fyzických oddílů.

Největší změnou oproti předchozímu skriptu je, že automat některé získané hodnoty ukládá do JS objektu, který nakonec každého *Script state* uloží do vyhrazené proměnné jako JSON (pojmenovaného *'overview'* – přehled). Před spuštěním dalšího příkazu je rovněž vhodné provést kontrolu, že nedojde ke skenu již skenováné VG. To automat provádí ověřením, že: *je skenován sekundární DD* a *primární DD je nastaven* (není *'sysdumpnull'*) a *VG právě skenovaného VG je stejná jako VG primárního DD*.

Pokud jsou všechny tři podmínky platné, tak spouštění následujícího příkazu není nutné. V opačném případě je připraven a spuštěn příkaz pro sken aktuálně zjištěné VG.

```
$ lsvg rootvg:
VOLUME GROUP:   rootvg      VG IDENTIFIER:  00e64...e6591
VG STATE:       active      PP SIZE:        128 megabyte(s)
VG PERMISSION:  read/write  TOTAL PPs:      510 (65280
   megabytes)
MAX LVs:        256         FREE PPs:       81 (10368 megabytes)
LVs:            32          USED PPs:       429 (54912
   megabytes)
OPEN LVs:       30          QUORUM:         2 (Enabled)
TOTAL PVs:      2           VG DESCRIPTORS: 3
...
```

Výpis č. 4.8: Zkrácený výstup příkazu lsvg [Vlastní zpracování]

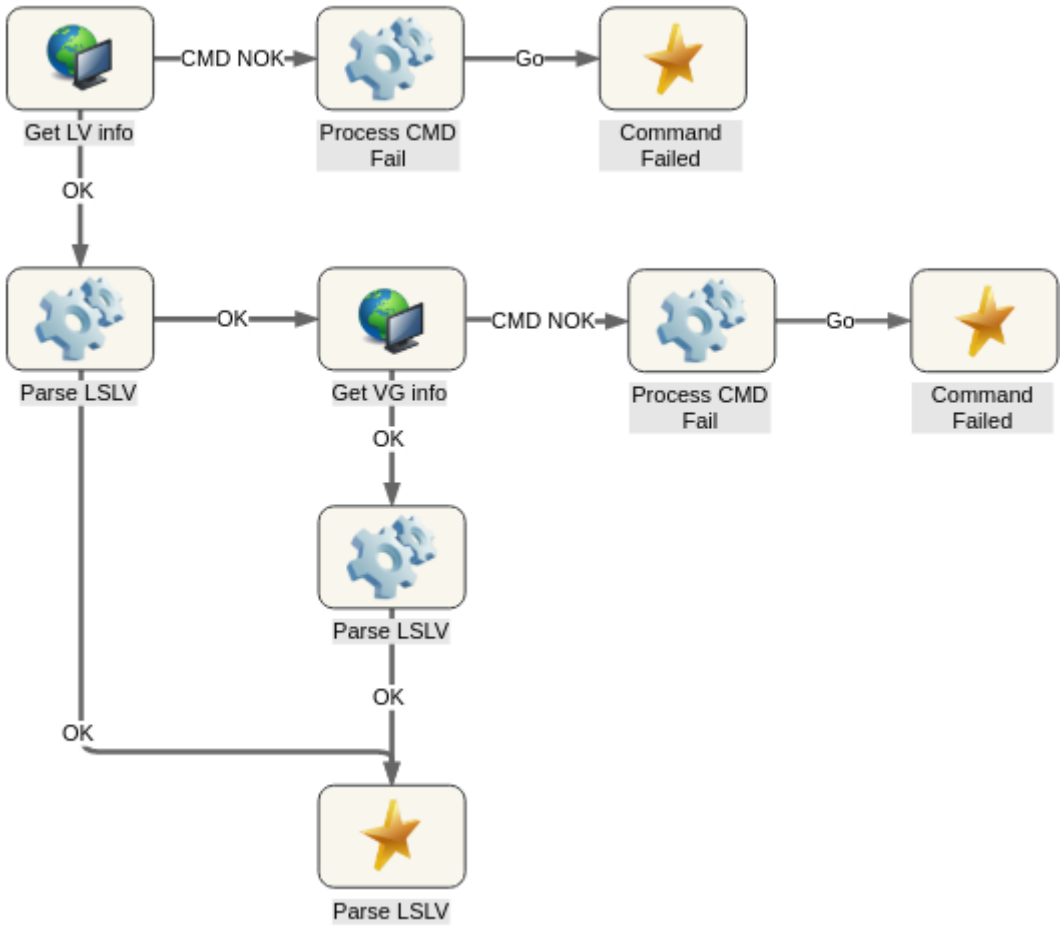


Obrázek č. 4.2: Fungování komponenty pro sken dump device [Vlastní zpracování]

Výstup příkazu lsvg je opět zachycen, a zpracován dalším skriptem. Užití příkazu a jeho výstup je ukázán na výpisu 4.8. Z tohoto výpisu opět získávám hodnoty potřebných parametrů, pomocí regulárních výrazů. Pro další práci jsou stěžejní tyto údaje:

- *PP SIZE* – velikost fyzického oddílu,
- *jednotka PP SIZE* – jednotka velikosti fyzického oddílu,
- *TOTAL PPs* – celkový počet PP ve VG,
- *FREE PPs* – počet volných oddílů,
- *TOTAL PVs* – počet používaných disků.

Po zpracování výstupu příkazu lsvg dojde k uložení zjištěných údajů (opět do *overview*). Poté je skript i automat *Scan Dump Device* ukončen, a dále se zpracovává hlavní automat. V něm může dojít k opětovnému volání této komponenty (pokud je třeba zjistit stav sekundárního DD).



Obrázek č. 4.3: Automat Scan DD [Vlastní zpracování]

4.1.4 Hlavní tok programu – dokončení diagnostiky

Pro dokončení diagnózy je ještě třeba provést tři příkazy. Prvním z nich je zjištění volného místa v copy directory. K tomu účelu používám příkaz *df* s parametrem *-m*, který mění výstup příkazu *df* na výpis v megabytech tak, jak ukazuje první část výpisu 4.9.

```
$ df -m /var/adm/ras
Filesystem  MB blocks   Free  %Used  Iused  %Iused  Mounted on
/dev/lv11      16.00   15.46     4%     18      1%  /var

$ df -m /var/adm/ras | tail -n 1 | awk -F" " '{print $3"_"$7}'
15.46_/var
```

Výpis č. 4.9: Zjištění volného místa pro copy directory [Vlastní zpracování]

Druhý (delší) příkaz ukazuje, jakým způsobem tento výstup server pro automat zpracuje, pomocí *roury*. Příkaz *'tail -n 1'* z příkazu vrátí pouze poslední řádek (a ne hlavičku). *Awk* používám proto, abych z výpisu vrátil pouze třetí a poslední (sedmý) sloupec. Třetí sloupec totiž obsahuje množství volného místa v daném souborovém systému, což je také velikost dostupného místa v copy directory. Sedmý sloupe poté ukazuje tzv. *mount point*, což je cesta k danému souborovému systému.

Poslední dva příkazy, které jsou v této fázi potřeba spustit slouží ke zjištění místa, které DD potřebuje. Oba příkazy jsou ukázány na výpise 4.10.

```
$ sysdumpdev -e
$ errpt -a -j E87EF1BE | sed -n '/Largest dump device size/,$p'
```

Výpis č. 4.10: Příkazy pro zjištění velikosti případného dumpu [Vlastní zpracování]

První příkaz vypíše zprávu *'Estimated dump size in bytes:'* a odhadovanou velikost příštího dumpu v bytech. Druhý příkaz poté vypíše chybová hlášení o právě opravované chybě *E87EF1BE*, a z tohoto výpisu odstraní úvod vedoucí až k prvnímu (nejnovějšímu) stavu *dump device* a potřebnému místu.

Zpracování výstupů diagnosticky

Další JS skript poté zpracuje výstup všech tří příkazů. Parsování probíhá podobným způsobem jako v předchozích případech, a to pomocí regulárních výrazů. Výsledky všech příkazů jsou poté přidány do objektu *overview*. Tím ale práce tohoto skriptu nekončí. Automat má nyní údaje o velikostech svazků a potřebném místě, ale všechny údaje jsou v různých jednotkách. Proto je třeba všechny velikosti převést na společnou jednotku. Za tu jsem zvolil megabyte.

```
function bytesConvertMultiplier(unit) {
    powers = {
        'b': 0, 'k': 1, 'm': 2, 'g': 3, 't': 4
    }
    power = powers[unit.charAt(0).toLowerCase()];
    return Math.pow(1024, power);
};

function bytesConvert(amount, fromUnit, toUnit) {
    var fromMultiply = bytesConvertMultiplier(fromUnit);
    var toMultiply = bytesConvertMultiplier(toUnit);
    return amount * fromMultiply / toMultiply;
}
```

Výpis č. 4.11: Funkce pro konverzi datových jednotek [Vlastní zpracování]

K přepočtu používám JS funkce, které jsem pro tento případ napsal. Pomocná funkce *bytesConvertMultiplier* na základě prvního neprázdného znaku rozhodne, jaký multiplikátor (mocninu čísla 1024), bude v dalším kroku aplikován na vstupní množství. Po zjištění vstupního a výstupního multiplikátoru už jen hlavní funkce, funkce *bytesConvert* aplikuje oba multiplikátory na vstupní množství.

Po převedení všech uložených velikostí, automat může porovnat oba zjištěné odhady velikosti případného dumpu. Z obou velikostí je pro další práci vybrána větší hodnota. Ta je uložena do nového klíče v objektu *overview*, a oba původní klíče jsou smazány.

Jednoduchým porovnáním potřebného místa (odhadnuté velikosti dumpu) a volného místa v *copy directory* automat určí jestli je v této složce dost místa.

Zhodnocení velikosti DD

Posledním krokem v diagnostice je kontrola, které LV jsou příliš malé pro další dump. Kroky jsou opět stejné pro oba DD.

Kontrola *dump device* začíná zjištěním, jestli je zařízení zrcadlené. To lze zjistit porovnáním, jestli je počet LP stejný počtu PP. Pokud jsou obě hodnoty rozdílné (PPs > LPs), tak si automat vytvoří záznam o počtu kopií. V opačném případě LV není zrcadlený, a počet kopií (proměnná *lv.copies*) je nastaven na hodnotu 1.

Nyní je nutné zjistit, kolik místa v zařízení chybí. Tento údaj získám odečtením aktuální velikosti LV od předpokládané velikosti dumpu (počet LP * velikost PP). Pokud vyjde záporné číslo, tak je LV dostatečně velká. V opačném případě je nutné LV rozšířit. Pokud je DD třeba rozšířit, tak je ještě potřeba zjistit, jestli je proto ve VG dostatek místa,

a jestli po případné rozšíření LV nezabere více místa v dané VG, než kolik zákazník povolil v parametrech. Tento postup ukazuje výpis 4.12. Tím je diagnostická část automatu hotová.

```
function evaluateDD(dd) {
  // lv - údaje o LV, vg - údaje o VG
  // Kolik místa chybí [MB]?
  lv.missingSize = overview.dumpSize - lv.LPs * vg.PPsize;
  lv.needsExtension = (lv.missingSize < 0);
  if (lv.needsExtension) { // DD potřebuje rozšířit:
    // Kolik místa chybí [PPs]?
    lv.missingLPs = Math.ceil(lv.missingSize / vg.PPsize);
    var missingPPs = lv.missingLPs * lv.copies;
    var limitPPs = vg.totalPPs * s_max_DD_size_threshold / 100;
    if (vg.freePPs > missingPPs) { // VG je dost velká
      // ověření zákaznického limitu:
      lv.canExtend = (thresholdPPs >= missingPPs + lv.PPs);
    } else { // VG není dost velká:
      lv.canExtend = false;
}}};
```

Výpis č. 4.12: Funkce pro zhodnocení velikosti DD [Vlastní zpracování]

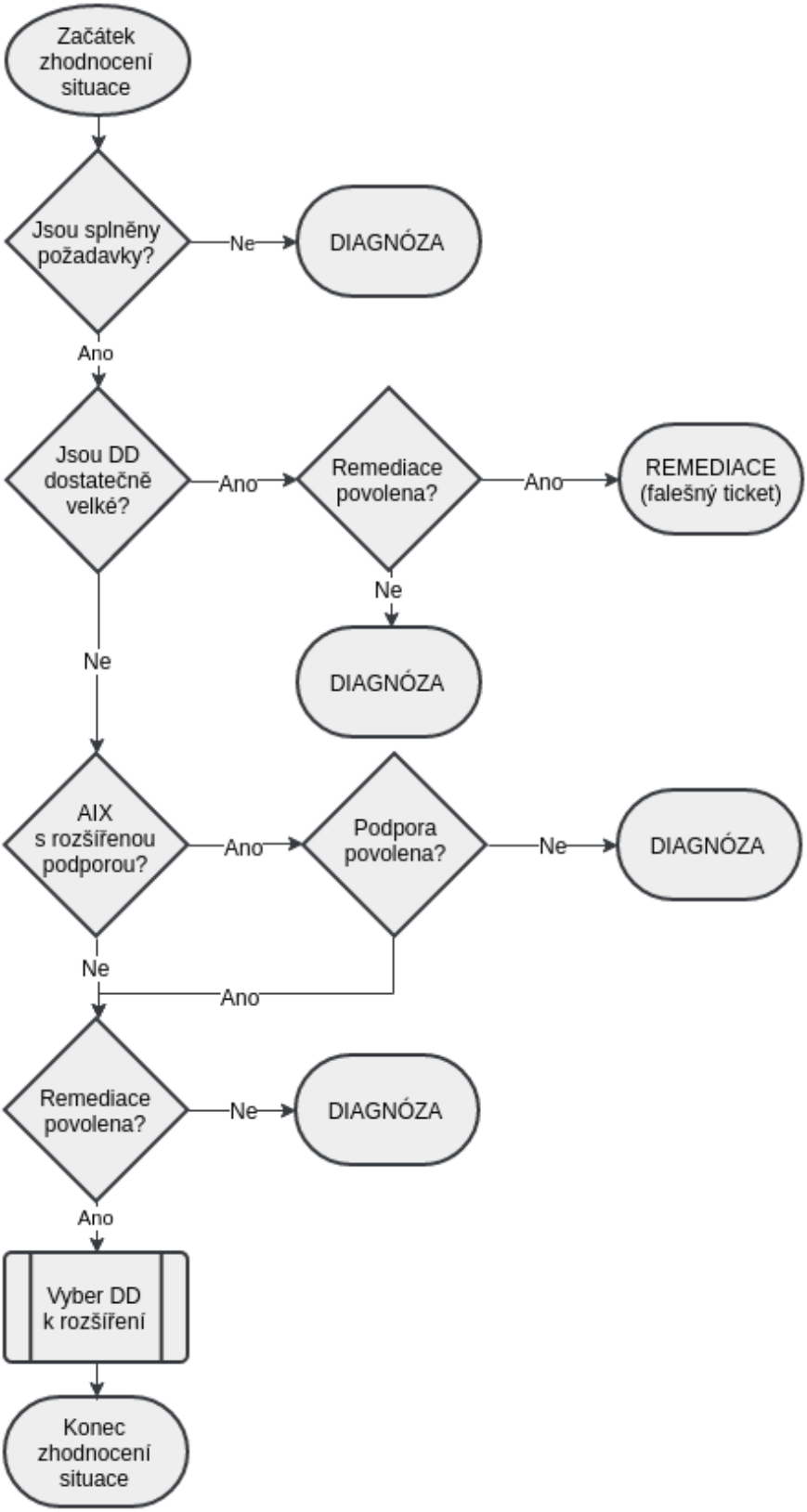
4.1.5 Hlavní tok programu – vyhodnocení stavu serveru

V tuto chvíli automat zná stav serveru. Před zásahem na serveru, je ale potřeba ověřit, jestli je to možné a v souladu s požadavky zákazníka. Proto je stav serveru porovnán se zákaznickým nastavením, jehož parametry byly vysvětleny v kapitole 4.1.1. Postup tohoto vyhodnocení je zobrazený na obrázku 4.4.

Zhodnocení požadavků na server

Zákazník má možnost nastavit vlastnosti pro daný server. Proto nastavení obsahuje parametr *s_requirements*, který v poli uchovává JSONy s nastavením, tak, jak to ukazuje výpis 4.2. Pro splnění tohoto testu musí alespoň jeden řádek – politika (zákazník jich může nastavit libovolný počet), anebo nesmí být nastavena žádná.

Pro kontrolu každého klíče jsem vytvořil samostatnou funkci, která daný fakt zkontroluje, a vytvoří případnou chybovou hlášku pro výstup programu. Následující seznam ukazuje všechny tyto parametry a princip jejich kontroly.



Obrázek č. 4.4: Vyhodnocení stavu serveru vůči nastavení [Vlastní zpracování]

- *checkTier()* – kontrola produkční úrovně serveru (proti záznamu ze CMDB). Dostupná je rovněž hodnota *'n/a'* (jakákoliv hodnota).
- *checkMonitoredAddress()* – provádí kontrolu, že adresa serveru je v určitém rozsahu IP adres (*'x'* v hodnotě parametru značí libovolné rozmezí). Funkce pomocí regExp získá všechny části adresy serveru, a poté je postupně porovná se zadanou maskou.
- *checkLvType()* – provede kontrolu, že všechny používané DD mají požadovaný typ. Umožňuje zadání hodnot *'sysdump'*, *'paging'* a *'n/a'*.
- *checkLvMirrors()* – kontroluje zrcadlení LV. Automat umožňuje vynucení zrcadlení (*'both'*), zákaz zrcadlení (*'none'*), jeden LV zrcadlený (*'one'*) anebo libovolný stav (*'n/a'*).
- *checkDdCount()* – kontroluje počet nastavených DD. Provádí se podle obsahu proměnných *primaryDD* a *secondaryDD* (hodnot zjištěných příkazem *'sysdumpdev -l'*. Dostupné hodnoty jsou stejné jako u předchozího parametru.
- *checkSameSize()* – tento parametr porovnává velikost nastavených DD. Ta může být vyžadovaná (*'required'*), zakázaná (*'forbidden'*) nebo libovolná. Výsledek této kontroly rovněž závisí na výsledku předchozí funkce. Například, pokud daný server smí používat pouze jeden DD, tak nesmí mít přikázanou stejnou velikost pro obě zařízení.
- *checkDdsOnSameVG()* – umožňuje vynucení nebo zákaz umístění obou DD na stejnou VG.

Zhodnocení dalších parametrů

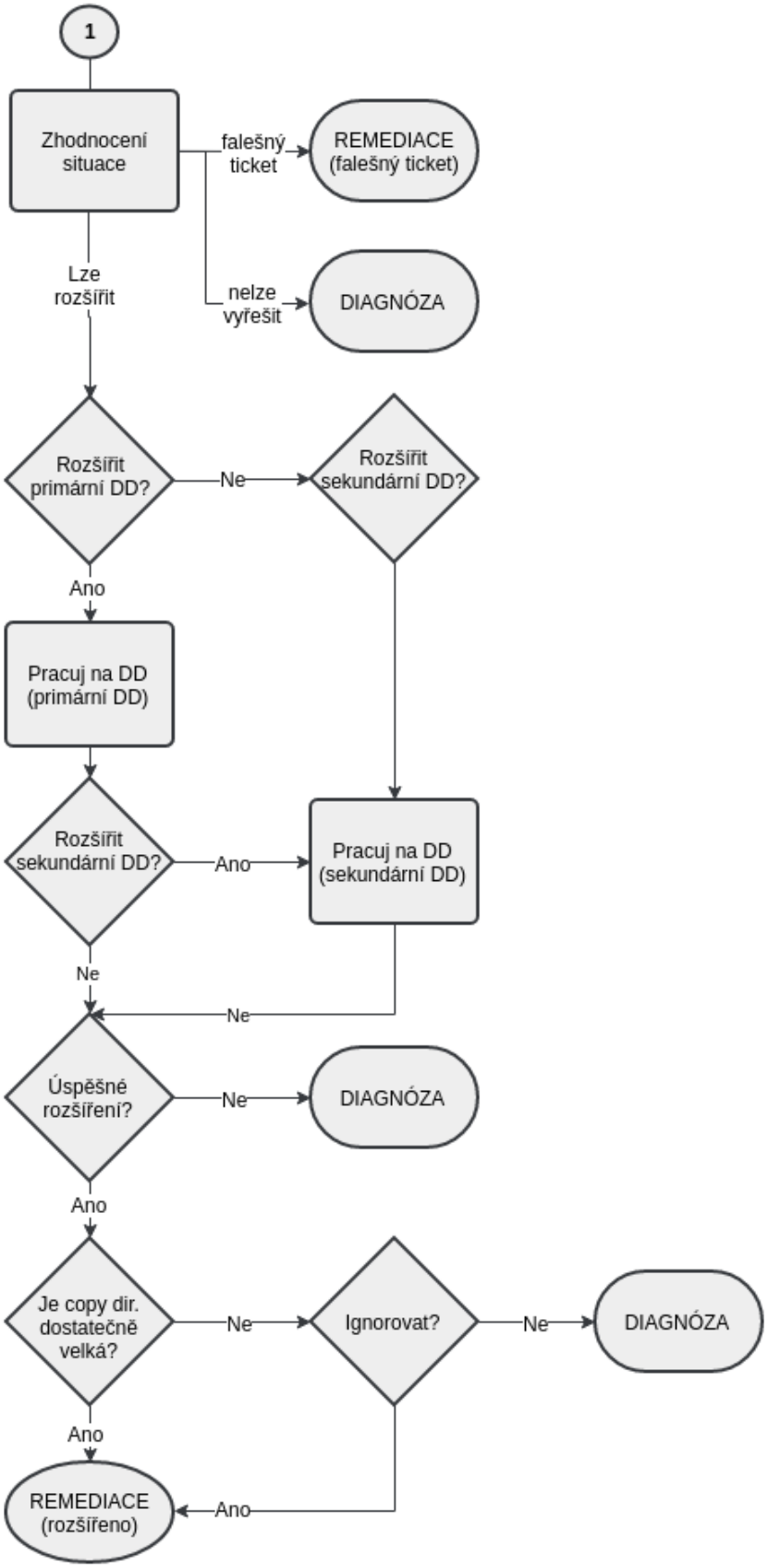
Poté dochází ke kontrole velikosti obou DD. Pokud by používaná zařízení byla dostatečně velká, tak může být ticket uzavřen, pokud to tedy nezakazuje parametr *s_diag_close_fake*. Ten se ale uplatňuje jen v diagnostickém módu.

Dále je provedena kontrola verze systému. Verze s rozšířenou podporou totiž nemusí být automaticky podporované, což udává parametr *s_remediate_older_AIX*. Poslední kontrolou je zapnutí diagnostického módu (*s_diagnosis_mode*).

Nesplnění požadavků z předchozí kapitoly nebo kteréhokoliv z právě zmíněných parametrů ukončí automat a eskaluje ticket s popisem důvodu k eskalaci a souhrnem stavu serveru.

Příprava na rozšíření

Pokud automat nebyl dosud ukončen, tak probíhá výběr DD k rozšíření, protože obě LV používaná jako DD potřebují rozšíření, a zásah na serveru je povolen.



Obrázek č. 4.5: Fungování Dump Device Extender – 2. část [Vlastní zpracování]

- Rozšíření obou LV:
  - pokud DD mají stejnou velikost a potřebují rozšíření,
  - pokud obě různě velká zařízení potřebují rozšířit a rozšíření obou zařízení je povoleno (parametr *s_extend_only_largest* je nastaven na *false* - vypnut).
- Rozšíření jednoho LV:
  - pokud DD nemají stejnou velikost, ale jen jedno zařízení potřebuje rozšířit,
  - pokud LV mají různou velikost, ale parametr *s_extend_only_largest* je nastaven na *true* (zapnut),
  - pro rozšíření byly vybrány oba DD, přičemž rozšíření prvního LV selhalo. Zapnutý parametr *s_continue_on_extend_fail* v takovém případě umožňuje rozšíření druhého DD.
- Nerozšíření žádného LV:
  - v případě selhání rozšíření,
  - pokud zásah není z jakéhokoli důvodu prováděn (viz. výše).

Po vybrání DD pro zvětšení je pro tato zařízení volána komponenta *Work on Dump Device*. Nutno dodat, že dokončení diagnózy a zhodnocení situace probíhá v jednom JS skriptu [6].

4.1.6 Komponenta Work on Dump Device

Poslední stěžejní částí automatu je komponenta, jejímž cílem je rozšíření konkrétního *dump device*. Fungování této komponenty je popsáno na diagramu 4.6.

Celá komponenta začíná skriptem pro přípravu příkazů, dle nastavení daného LV. Postup rozšiřování se totiž liší dle použitého souborového systému (*sysdump* nebo *paging*). V případě, že je nastaven svazek typu *sysdump*, tak skript pro rozšíření nastaví příkaz *extendlv*.

```
extendlv pdumplv 10
```

Výpis č. 4.13: Použití příkazu pro rozšíření LV [Vlastní zpracování]

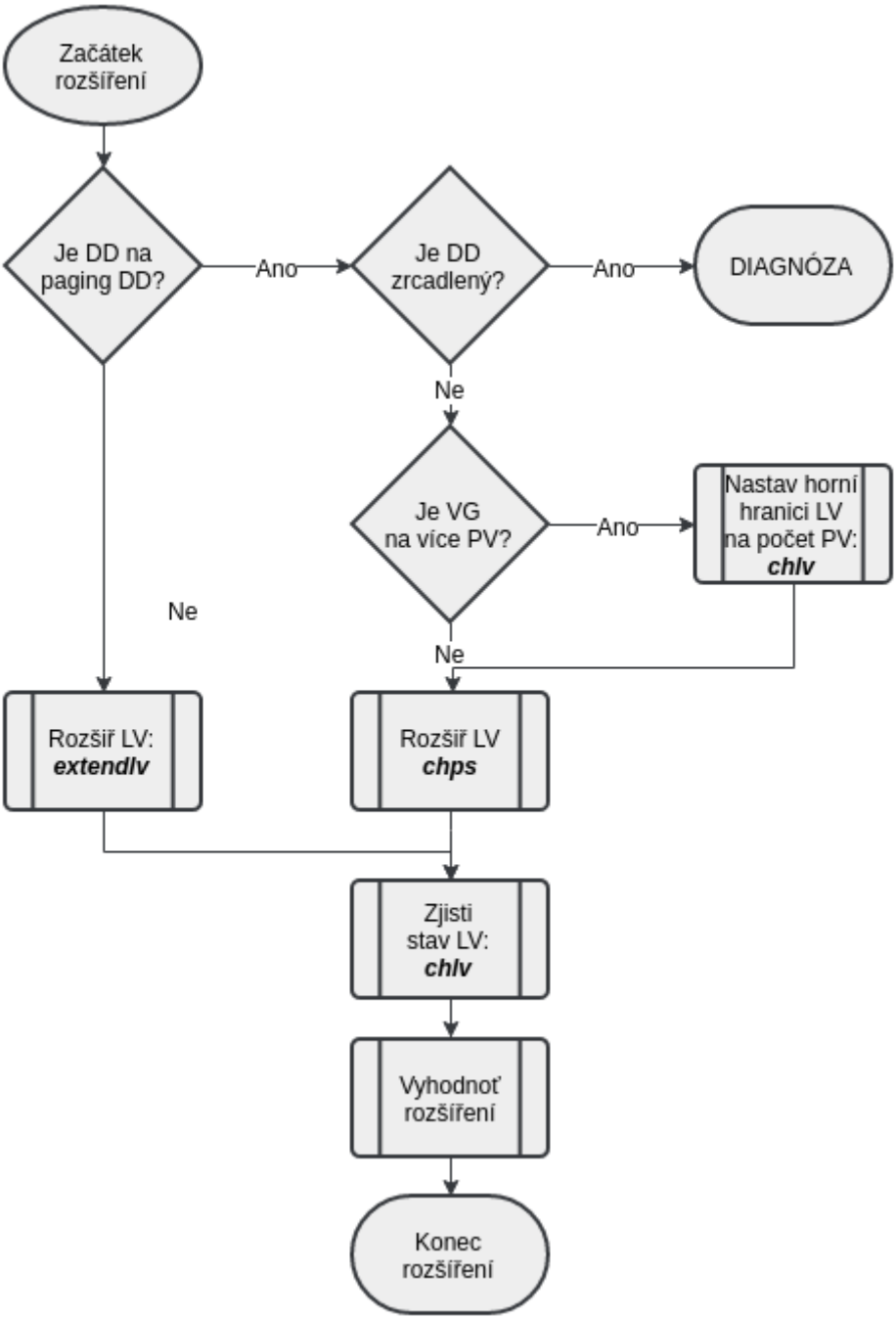
První hodnota v příkazu *extendlv* vybírá, který LV bude rozšířen. Následuje počet logických oddílů, který je potřeba přidat.

Rozšíření paging LV

Pokud je jako DD použit swapovací svazek, tak ještě probíhá kontrola, jestli je daný svazek zrcadlený. Pokud ano, tak k automatickému rozšíření nedojde[7]. Rovněž je zjištěno,

[6] Takže téměř celý obsah kapitol 4.1.4 a 4.1.5 tvoří v automatu jeden javascriptový stav.

[7] Na základě doporučení pro administrátory z kapitoly 3.2.5.



Obrázek č. 4.6: Fungování komponenty Work on DD [Vlastní zpracování]

jestli LV používá více disků. Tento údaj byl získán již během skenu DD, příkazem *lsvg* (položka *TOTAL PVs*). Pokud je použito více PV, tak bude potřeba použít ještě příkaz *chlv*.

```
chlv -u 2 pdumplv
chps -s 10 pdumplv
```

Výpis č. 4.14: Použití příkazů pro rozšíření swapovacího LV [Vlastní zpracování]

Výpis 4.14 ukazuje, jakým způsobem se oba potřebné příkazy používají. První z nich (*chlv*), mění charakteristiky LV. Parametr *-u* vybírá změnu hranice počtu PV pro rozšíření. Následuje upřesnění počtu disků, které budou použity pro rozšíření. Posledním parametrem je název LV.

Samotné rozšíření provede příkaz *chps*, který mění vlastnosti paging space. Použitý parametr *-s* slouží pro přidání místa, který určuje první hodnota. Poslední hodnota v příkazu opět určuje vybraný LV.

Kontrola rozšíření

Posledním příkazem, který úvodní skript této komponenty musí připravit, je opět příkaz *lsvg*. Ten automat použije k ověření výsledků rozšíření tak, jak to ukazuje výpis 4.15.

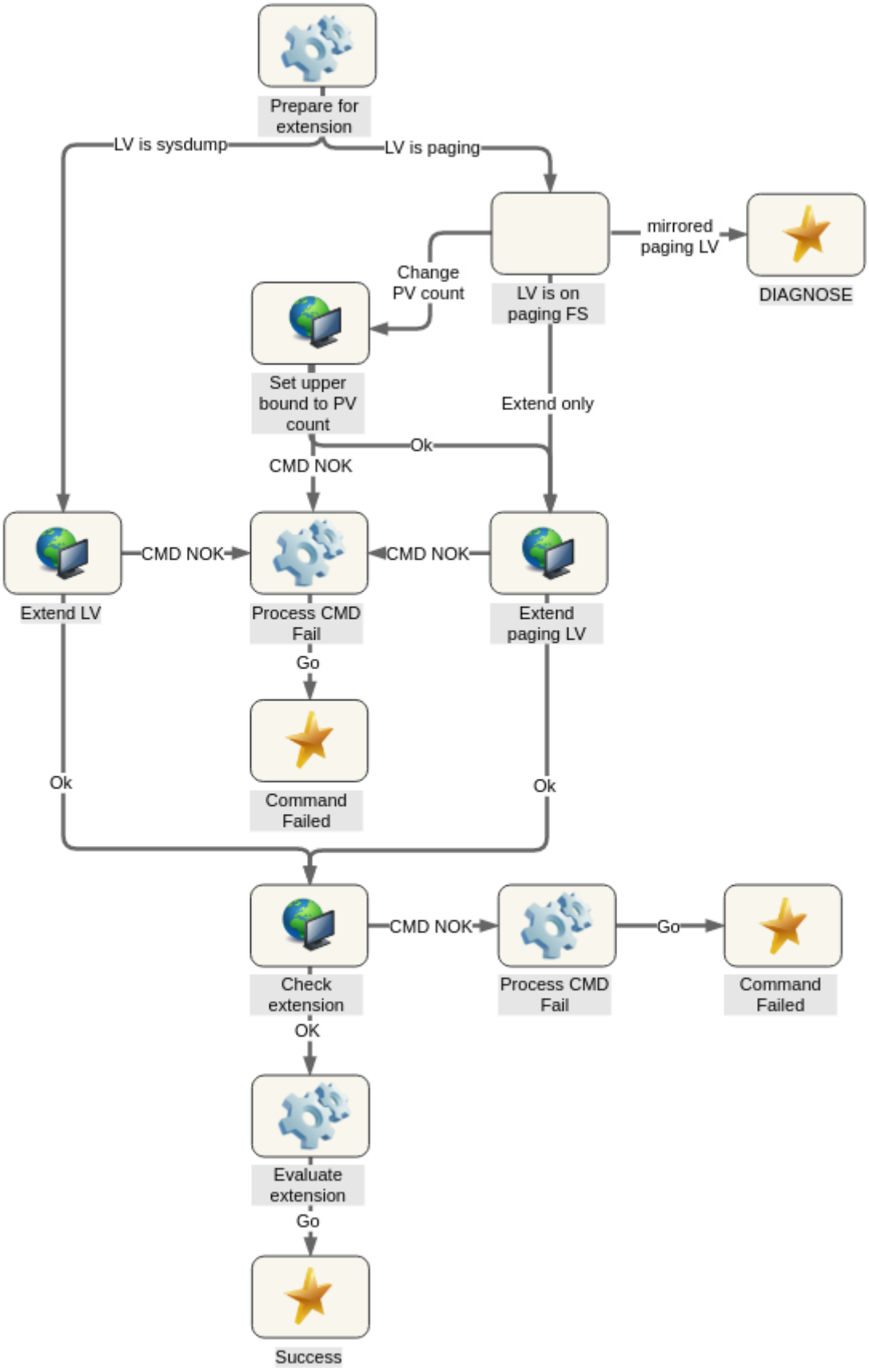
```
$ lsvg -l rootvg
rootvg:
LV NAME  TYPE     LPs  PPs  PVs  LV STATE    MOUNT POINT
...
pdumplv  sysdump  11   11   2    open/syncd  N/A
...

$ lsvg -l rootvg | awk -F" " '{print $1, $3}' | grep 'pdumplv'
pdumplv 11
```

Výpis č. 4.15: Příkaz pro kontrolu rozšíření LV [Vlastní zpracování]

Protože jsem opět použil *roury*, tak první část výpisu ukazuje použití samotného příkazu *lsvg* s parametrem *-l*, který vypíše seznam všech LV v zadaném VG. K příkazu jsem přidal příkaz *awk*, pro výběr sloupců, které automat potřebuje znát (jméno LV a nový počet LV). Druhý příkaz je *grep*, který vybere řádek obsahující zadaný text, nyní jméno rozšiřovaného svazku.

Po přípravě všech příkazů, a jejich provedení komponenta zkontroluje výstup posledního příkazu. Nová velikost LV je porovnána s požadovanou. Výsledek rozšíření je přidán do objektu *overview*, klíče *extensionOk*.



Obrázek č. 4.7: Automat Work on DD [Vlastní zpracování]

4.1.7 Hlavní tok programu – ukončení automatu

Po zvětšení nastavených zařízení je potřeba vyhodnotit výsledek činnosti automatu. To je provedeno na základě výsledků komponenty *Work on DD* (či nedokončení komponenty) a velikosti *copy directory*.

První kontrola je provedena pomocí kontroly klíčů *extensionOk* pro nastavené LV v objektu *overview*. Pokud nastavená zařízení ukazují hodnotu true, tak je rozšíření považováno za úspěšné. V opačném případě je ticket opět doplněn o výstupy diagnózy a předešlé činnosti a eskalován.

Kontrola *copy directory* je posledním krokem. Pokud v adresáři není dostatek místa, tak je ještě kontrolován parametr *s_remediate_small_cpFldr*. Jeho zapnutí umožní zavření ticketu i v tomto případě.

Pokud vše proběhlo správně, tak automat sestaví hlášení o předchozím a novém stavu serveru, uzavře ticket, a předá řízení zpátky do wrapperu.

4.1.8 Postup při selhání příkazu

Možných důvodů pro selhání příkazu je více. Nejčastějším důvodem je výpadek připojení, ať už úplný nebo dočasný (timeout). Proto je třeba, aby po každém stavu byla provedena kontrola, že příkaz byl úspěšně dokončen.

Z každého stavu *Host Command* tedy vedou nejméně dva přechody. Každý kontroluje výstup příkazu. Jeden na úspěch (ten pokračuje na další stav v toku automatu). Druhý přechod je aktivován neúspěchem příkazu a vede do skriptu, který tuto situaci zpracuje a ticket eskaluje.

Tento skript z již získaných údajů sestaví hlášení o stavu serveru. Na začátku tohoto zápisu bude popis problému obsahující jméno stavu, použitý příkaz, číslo chyby a popis této chyby.

4.1.9 Sestavení hlášení pro aktualizaci logu ticketu

Automat může skončit mnoha různými způsoby, ale před předáním řízení zpátky do wrapperu vždy sestaví hlášení pro zákaznický service desk. Toto hlášení musí jednou větou shrnovat výstup automatu, a poté podrobněji rozepsat stav serveru. Proto jsem napsal jednoduchý skript, který toto hlášení sestaví.

Aktualizace ticketu začíná stručným shrnutím situace, které je nastaveno prováděným skriptem podle zakončení automatu. Úvodní hlášky jsou popsané v následující sekci.

Toto shrnutí situace je hned rozvedeno podrobnějším popisem situace. Pokud selhal příkaz, tak je tímto důvodem již popsaný důvod selhání příkazu. Jiným důvodem může být diagnostické ukončení, jehož důvod bude rovněž důkladněji popsán. Zpráva o rozšíření DD nebo důležitá hlášení jsou rovněž zmíněna v této sekci.

```
Largest dump device(s) was extended to required size.

Primary dump device was extended by 10 PPs and is now big enough.
Secondary dump device is not set.
Copy folder is too small to contain another dump!

Status of dump devices:
- Estimated size of next dump is: 451 MBs.
- Primary dump device is set on 'pdumplv':
  type 'sysdump',
  not mirrored,
  currently uses 11 PPs, 11 LPs (PP size: 128 MBs),
  before used: 1 PPs, 1 LPs,
  is located on 'rootvg':
    total of 510 PPs (free: 81 PPs),
    Total PVs: 2.
- Secondary dump device is not set.
- Copy directory is set on 'pdumplv' in /var/adm/ras:
  set on LV: '/var',
  has 15.46 MBs free.
```

Obrázek č. 4.8: Ukázka výstupu aktualizace logu ticketu [Vlastní zpracování]

Poslední sekce obsahuje výchozí stav serveru. Ten je sestaven na základě objektu *overview*. Každý z klíčů tohoto objektu je před svým voláním kontrolován, protože ještě nemusel být vytvořen. Stav serveru obsahuje:

- odhadnutou velikost dumpu,
- stav obou DD:
  - funkce DD a název LV,
  - typ LV,
  - nastavení zrcadlení,
  - výchozí a novou velikost LV, velikost jedné PP,
  - odhadovaný počet chybějících (přebývajících) LP,
  - jméno VG,
  - počet PP (celkový i volný)
  - počet použitých disků,
- stav *copy directory*.

Možná zakončení činnosti automatu

Při návrhu nového programu je důležité vědět, jaké výstupy může daný program mít. První seznam obsahuje situace, kdy byl ticket automaticky vyřešen.

- *Dump Device is big enough* - rozšíření nebylo třeba.
- *DD(s) was successfully extended* - automaticky rozšířené DD.

Následující seznam obsahuje situace, které skončí diagnózou. Tyto výstupy mohou být způsobené nastavením ticketu nebo nepodporovanými situacemi.

- *No Dump Device is set* - není nastavený žádný DD.
- *Remediation not allowed* - pro diagnostický mód.
- *Remediation not allowed on this AIX version* - pokud není povolena podpora verzí AIX ve fázi rozšířené podpory.
- *Requirements were not met* - požadavky zákazníka na server nebyly splněny.
- *Extension of paging mirrored DD is not supported* - automatické rozšíření zrcadlené LV typu paging není automatem podporováno.
- *Copy directory is too small* - aktivována, pokud došlo k rozšíření DD, ale copy directory nemá dostatek místa a uzavření ticketu není povoleno.
- *Insufficient free space* - pro případ nedostatečného místa ve skupině svazků nebo překročení limitu pro rozšíření.
- *Extension was not successful* - aktivována nedostatečným rozšířením DD.

Poslední soupis obsahuje možné chybové výstupy automatu, pro případ libovolného selhání.

- *Connection Error* - wrapperu se nepodařilo připojit.
- *Unsupported OS* - OS nebo verze AIX není podporovaná.
- *Unknown Error* - výchozí výstup automatu pro případ neznámé chyby.
- *Execution of host command failed* - selhání příkazu.

4.1.10 Shrnutí

V této kapitole prošel implementovaný automat *Dump Device Extender*, jehož úkolem je remediace chyby operačního systému AIX *'The largest dump device is too small'*.

Automat po spuštění (a připojení se na server) provede diagnózu nastavení dostupných DD (pomocí komponenty pro skenování DD), zjistí odhadovanou velikost dumpu a zjistí dostupné místo v *copy directory*. Po dokončení diagnózy automat provede vyhodnocení situace a aplikuje zákaznické požadavky. Poté provede rozšíření vybraných logických svazků.

Automat umožňuje rozšíření obou DD a to nejen v případě, že obě zařízení jsou stejně velká. Rovněž existuje možnost podpory verzí, které jsou jen ve fázi rozšířené podpory. Nastavení upravuje chování automatu v mnoha situacích. Jedním z těchto parametrů je tzv. diagnostický mód, kdy automat nebude provádět žádné zásahy na serveru. Další skupina parametrů umožňuje kontrolu nastavení dumpu na daném serveru. Automat rovněž podporuje práci s příkazem sudo. Tímto považuji všechny požadavky z kapitoly 3.4 za splněné.

4.2 Výhledy do budoucna

S dokončením první verze automatu Dump Device Extender aktivní vývoj tohoto automatu nekončí. Jako vždy, i v tomto programu existuje prostor pro další funkcionalitu. Nyní tuto možnou funkcionalitu nastíním.

Nové parametry

Vzhledem k pružnosti parametrizace tohoto automatu (díky použití formátu JSON) přidání nových parametrů nevyžaduje zásadní změny programu. Pouze připsání procedury, která daný parametr v potřebnou chvíli vyhodnotí.

Jedním z těchto nových parametrů může být *rezerva*. Ta by automatu umožnila zvětšit LV o víc místa, než je nutné (oproti současnému návrhu, kdy je přidáno pouze tolik místa, kolik je třeba pro uložení současného dumpu). Parametr by v procentech udával velikost rezervy oproti současné velikosti dumpu. To může snížit počet budoucích výskytů tohoto problému.

Automatická rekonfigurace dump device

Další možností pro rozšíření funkcionality může být automatická rekonfigurace dump device. Ta by server překonfigurovala podle nejnovějších standardů. Důvodem pro tuto eventualitu může být zrcadlení *paging* LV, které v tomto automatu nejsou podporovány. Nová komponenta by v rámci parametrem definovaných VG vytvořila nové logické svazky typu *sysdump*. Do těch by poté přesměrovala dump.

Vzhledem ke složitosti této procedury může být tato funkcionalita rovněž implementována ve zcela samostatném proaktivním programu, který by ale byl spouštěn z automatu *Dump Device Extender*.

4.3 Ekonomické zhodnocení

Nyní nastal čas implementované řešení ekonomicky zhodnotit. To provedu pomocí metod Doby návratnosti a ROI. Uvažovanou jednotkou bude čas. Pro tuto analýzu předpokládejme, že čas investovaný do vývoje řešení a čas pracovníků service desku je stejně cenný.

Náklady

Největším nákladem je v tomto případě čas strávený vývojem řešení. Ten jsem vyčíslil na 210 člověkohodin včetně nasazení automatu pro počáteční soubor serverů. Pro další údržbu a práci týmu budu počítat 25 % ze získaného času (přínosu).

Přínosy

Automat *Dump Device Extender* je v době tvorby této analýzy již 4 měsíce nasazen pro přibližně jednu polovinu serverů z analyzovaného prostředí (přibližně 1500 serverů). Tabulka 4.1 ukazuje, kolik zásahů automat v tomto období provedl. Sloupec *Času ušetřeno na ticket* ukazuje odhadovaný ušetřený čas, a sloupec *Celkem ušetreno času* poté ukazuje celkový čas, který automat ušetřil. Oba uvedené časy jsou v hodinách.

Tabulka č. 4.1: Výstupy automatu ve sledovaném období [Vlastní zpracování]

Výstup	Počet ticketů	Času ušetřeno na ticket [h]	Celkem ušetřeno času [h]
Connection Error	8	0.00	0.00
Unknown Error	1	0.00	0.00
Execution of hostcommand failed	7	0.00	0.00
Remediation not allowed	131	0.17	21.83
Requirements were not met	39	0.17	6.50
Insufficient free space	35	0.17	5.83
Extension was not successful	1	0.17	0.17
Copy directory is too small	1	0.25	0.25
Dump Device is big enough	4	0.50	2.00
DD(s) was successfully extended	77	0.50	38.50
Celkem	304	-	75.08

Při určování času na vyřešení incidentu jsem vycházím z předpokladu, že na manuální vyřešení jednoho ticketu na tuto chybu (*E87EF1BE*) je počítáno 30 minut. Ušetřený čas na incident jsem poté stanovil pro hlavní výstupní situace následovně:

- automat nepomohl service desku: 0 minut,
- automat sestavil diagnostický report: 10 minut,
- automat nemohl zavřít ticket, ale problém vyřešil: 15 minut,
- automat problém vyřešil a zavřel ticket: 30 minut.

Automat má plánovanou životnost 4 roky. Přínosy automatu budu tedy počítat pro 4 roky se stávající výnosností automatu. Automat byl ve sledovaném období nasazen pouze pro část celého prostředí, a i nadále dochází k rozšiřování jeho nasazení. Proto považuji výsledky následujících ukazatelů za pesimistické odhady.

4.3.1 Doba návratnosti investice

Dobou návratnosti investice (v letech) jsem určil pomocí následujícího vzorce. V něm počítám s ročním ziskem a hodnotou investice. Hodnota investice jsou již zmíněné náklady (210 hodin). Roční zisk získám přepočtem ušetřeného času za sledované období (75 hodin za 4 měsíce) na délku jednoho roku.

$$Doba\ navratnosti = \frac{rocni\ zisk}{investice} = \frac{\frac{zisk\ za\ obdobi}{delka\ obdobi} * 12}{investice} \tag{4.1}$$

Investice se dle zjištěných hodnot vrátí do jednoho roku od uvedení do provozu (0.93 roku).

4.3.2 ROI

K určení hodnoty ukazatele Return On Investment jsem použil následující vztah. Celkový zisk jsem určil pomocí plánované délky investice (4 roky) a zisku za jeden rok, který jsem získal stejným způsobem jako v předchozím výpočtu (*rocni zisk * delka investice*).

$$ROI = \frac{celkovy\ zisk - investice}{investice} * 100 \tag{4.2}$$

Ukazatel ROI vyšel 329.05 %. Jinými slovy se investice vrátí během své životnosti více než třikrát. Tento ukazatel ale nebere v potaz náklady na údržbu investice.

Čistou návratnost investice zjistím, když ukazatel ROI očistím o náklady na údržbu (25 % ze zisku). Tento ukazatel mi vyšel 246.79 %. Investice se tedy během svého trvání ušetří téměř 250 % času věnovaného na vývoj.

$$Cista\ navratnost = ROI - ROI * naklady\ na\ udrzbu \tag{4.3}$$

4.3.3 Další přínosy

Každá investice má rovněž přínosy, které nelze přímo vyčíslit, a tak je tomu i v případě správy ICT prostředí pomocí automatických nástrojů.

Prvním velkým přínosem je standardizace správy a s tím související zvýšení kvality, což dokazuje i implementovaný automat pro rozšíření *dump device*. Navržená rutina vychází z mnoha doporučení, která IBM (jakožto tvůrce OS AIX) dává svým uživatelům. Zásahy na serverech poté probíhají vždy stejným způsobem. Automat rovněž umožňuje hlášení špatně nastaveného serveru, což service desku umožňuje nastavení serveru opravit.

S tím souvisí i zvýšení rychlosti zásahu na serverech. Jak již bylo napsáno, na opravu zkoumaného problému service desky obvykle počítají 30 minut. K době potřebné pro zásah na server ještě musíme připočítat čas potřebný pro reakci operátora na ticket. Automatická řešení potřebují na stejný zásah několik desítek vteřin až několik minut, podle vytíženosti infrastruktury. A také v případě, že zásah automatu skončil diagnostickým výstupem, může automat poskytnout stručný přehled o stavu serveru, který administrátorům může ušetřit významnou část práce.

Dále nesmíme zapomínat na zvýšení spokojenosti zaměstnanců. Opakování řešení stále stejných úkonů může vést nejen ke snížení pozornosti a zvýšení počtu chyb, ale také k poklesu spokojenosti. S implementací systémů, jako je IPC se zaměstnanci mohou věnovat řešení složitějších problémů nebo problémů, které automatické rutiny nezvládnou vyřešit.

Všechny již zmíněné přínosy vedou ke stejnému cíli, kterým je zvýšení kvality služeb a tedy i zmenšení dopadů na zákazníka (zkrácení doby výpadků služeb atd.). V neposlední řadě rovněž dochází ke snižování nákladů firmy, a její ICT prostředí se stává snáze škálovatelným. To vše přispívá ke zvýšení konkurenceschopnosti firmy.

4.4 Využití dalších řešení při správě ICT prostředí

V této práci jsem se věnoval ukázce praktického využití reaktivní automatizace řešení problémů na serverech. Jak jsem ale zmínil v kapitole 2.13, tak existují další dva druhy zásahů na serverech – proaktivní a prediktivní.

Automatické proaktivní zásahy jsou v dnešní době běžné. Čítají vše od skenů stavu sítě, po automatické nasazování serverů. Zjednodušenou ukázku proaktivní automatizace ukazuje i automat *Dump Device Extender* v případě, kdy server nesplňuje určité požadavky.

Nedochází sice k automatickému řešení problému, ale service desk je na danou situaci upozorněn, a může situaci napravit předtím, než způsobí problém.

Prediktivní zásahy dnes spočívají ve skenování stavu infrastruktury a aplikování získaných poznatků pro předvídání jejího budoucího stavu. Příkladem využití těchto skenů je odhalení selhávajícího hardware před tím, než se dané selhání vůbec projeví. Ve výsledku se tedy jedná o aplikaci metod data miningu a strojového učení na IT infrastrukturu.

Technologie strojového učení

Výběrem rutiny dle příznaků daného problému nebo upozorněním na budoucí problémy, ale využití zmíněných technologií nekončí. Jednou z možností pro budoucí systémy jsou i technologie, jako je systém IBM Watson. Ten poskytuje několik desítek způsobů pro analýzu dat a tvorbu vlastních aplikací s využíváním technologií pro zpracování obrazu, dokumentů a rovněž přirozeného jazyka, a to pomocí metod strojového učení.

Například služby pro zpracování přirozeného jazyka jsou již dnes součástí chytrých telefonů (asistentka pro iPhone Siri), operačního systému Windows (Cortana) nebo služba Alexa od firmy Amazon. Podobné služby se uplatňují rovněž při komunikaci zákazníka s helpdesky. Ať v čistě textové formě (tzv. chatbot) nebo i s hlasovou interakcí.

Využití těchto služeb poté v praxi vypadá tak tak, že zákazník chce provést úpravu poskytovaných služeb (např. zvětšit emailovou schránku) a kontaktuje helpdesk (telefonicky). Helpdesk (robot) s ním poté komunikuje pomocí služeb pro konverzaci (chatbot), převodu mluvené řeči na text (napřík *speech-to-text*) a textu na řeč (*text-to-speech*). Pomocí předdefinovaného dialogu (řízeného robotem) je identifikován požadavek zákazníka. Robot poté zjistí současný stav zákaznického systému a zahájí kroky potřebné k vyřešení požadavku. Zákazník je na konci zásahu o výsledcích informován stejným způsobem, jakým kontaktoval helpdesk.

Další metody automatizace

Při výčtu současných a budoucích technologií pro správu ICT nesmím opomenou ani *RPA* a *server build*. *RPA* (*Robotic Process Automation*) má blíže k běžné práci uživatele než většina zde popsaných řešení. Úkolem těchto nástrojů je totiž automatické provádění činností v GUI namísto uživatele. Toho se využívá převážně v programech, které nemají textové rozhraní, a kde je třeba opakovaně provádět stejnou činnost.

Server build neboli stavba serverů znamená automatizování instalace serverů (SW ne HW). Toho se využívá hlavně v cloudových službách, kde společnosti jako Amazon,

Microsoft nebo IBM poskytují svým zákazníkům, ať už firmám nebo jednotlivcům, možnost provozu vlastních serverů (tedy Infrastructure as a Service). Zákazník si zvolí vlastnosti požadovaného serveru, a ten je podle požadavků vytvořen. Samotná instalace operačního systému může proběhnout vytvořením (zkopírováním) a parametrizací virtuálního OS. Instalace dalších aplikací poté může být provedena pomocí proaktivních automatizačních nástrojů (například Chef). Tyto postupy lze také využít uvnitř firemní infrastruktury, anebo v případě prostředí, jaké jsem popsal v kapitole 3.1.

Integrace platforem

Dalším stavebním kamenem budoucích řešení musí být také vzájemná integrace jednotlivých nástrojů. To vyplývá již z předchozích příkladů. Robot helpdesku přímo nepřistupuje na koncové zařízení k postižené službě. K tomu účelu jsou používané jiné specializované nástroje. Bez propojení těchto systémů, ale robot nebude moci zjistit stav aplikace pro zákazníka, anebo splnit požadavek na vytvoření nového serveru. A obráceně. Nástroj, jako je IPC, nemusí přímo kontaktovat zákazníka a komunikovat s ním. Pouze přes integrační nástroje (obecně známé jako Application Integration SW) zašle požadavek na robota help desku.

Bezpečnost

Při návrhu a implementaci zde zmíněných technologií rovněž nesmí být opomenuta ani bezpečnost. Ta již mnohdy nemůže být zajištěna pouze za užití současných standardních technik bezpečností informací. S dalším rozvojem bezpečnosti mohou pomoci i zde zmíněné nástroje a techniky. Odstřihnutí lidského faktoru od řešení rutinních problémů nebo standardizace použitých postupů totiž nenapomáhají jen zvyšováním kvality poskytovaných služeb. Při správné implementaci mohou skokově zvyšovat schopnost firmy zvládat bezpečnostní audity (automatickou tvorbou a analýzou logů ze serverů). Nebo mohou být využity v plánech Disaster Recovery (například rychlou, automatickou obnovou služby při jejím fatálním narušení). Rovněž, ale nesmí být opomenuta bezpečnost i samotných automatizačních systémů.

Ať už je automatizace správy ICT systémů použita jakkoliv, výsledkem by měl být modulární, škálovatelný a flexibilní systém. Systém, jehož zákazník si může sám kdykoliv určit, jaké služby od daného systému požaduje, a příslušné změny vidět v co nejkratším čase. Takový systém svého zákazníka rovněž nesmí omezovat v jeho činnosti ani v případě chyb nebo poruch. Těmto řešením se již v dnešní době říká Automation as a Service (automatizace jako služba). A právě praktické ukázce implementace automatizace jako služby jsem se věnoval v této práci.

ZÁVĚR

Cílem této práce bylo ukázání přínosů automatizace při řešení problémů správy ICT prostředí. Problém, který jsem v práci zabýval je méně častá chyba operačních systémů AIX. Implementovaným řešením je automat Dump Device Extender, který jsem napsal v nástroji IPC.

V první části práce jsem proto popsal důležitá teoretická východiska práce. Věnoval jsem se hlavně unixovým systémům, ale také programovacím jazykům Java a JavaScript nebo základům správy ICT prostředí a monitorovacím systémům.

Poté jsem pokračoval analýzou operačního systému AIX, a řešeného problému, kterým je chyba *'The largest dump device is too small'*. Nedílnou součástí této kapitoly byl rovněž rozbor hlavních vlastností nástroje IPC, včetně jeho zapojení do stávající infrastruktury.

Hlavní část práce tvoří popis již zmíněného automatu. Při popisu jsem ukázal některé důležité části řešení, a to včetně ukázek kódu, výstupů jednotlivých příkazů a vývojových diagramů.

Závěrem celé práce poté bylo ekonomické zhodnocení vyvinutého programu, které jsem provedl na základě reálných výsledků automatu. Zde musím zdůraznit, že ačkoliv se jedná o méně častý problém a automat byl během sledovaného období používán pouze pro část zkoumaného prostředí, přesto se tato investice vrací během prvního roku.

Tímto považuji hlavní cíl práce a rovněž veškeré dílčí cíle práce za splněné.

SEZNAM POUŽITÉ LITERATURY

[1] OZ, Effy. *Management information systems*. 6th ed. Boston, Mass.: Thomson/ Course Technology, c2009. ISBN 978-142-3901-785.

[2] What is Middleware? *Wayback Machine* [online]. St. Petersburg FL: Defining Technology, ©1997-2008 [cit. 2017-03-10]. Dostupné z: web.archive.org/web/ 20120629211518/http://www.middleware.org/whatis.html

[3] *AIX for System Administrators* [online]. 2015 [cit. 2017-02-25]. Dostupné z: aix4admins.blogspot.cz

[4] CRAIG, Iain D. *Formal Models of Operating System Kernels*. Springer Science Business Media, 2007. ISBN 978-1-84628-718-3.

[5] *The Open Group* [online]. ©1995-2017 [cit. 2017-03-10]. Dostupné z: opengroup. org

[6] What Is UNIX? *Rochester Institute of Technology* [online]. Rochester NY, 2014 [cit. 2017-02-20]. Dostupné z: www.cis.rit.edu/class/simg211/unixintro/Whatis.html

[7] Using Unix. *Stanford University: School of Earth, Energy Environmental Sciences* [online]. Stanford, California: Stanford University [cit. 2017-03-14]. Dostupné z: earthsci.stanford.edu/computing/unix/

[8] SALUS, Peter H. *A quarter century of UNIX*. Reading, Mass.: Addison-Wesley Pub. Co., c1994. ISBN 02-015-4777-5.

[9] A Backgrounder on IEEE Std 1003.1. *The Open Group* [online]. IEEE and The Open Group [cit. 2017-03-12]. Dostupné z: www.opengroup.org/austin/papers/ backgrounder.html

[10] *IBM Knowledge Center* [online]. IBM, 2015 [cit. 2017-02-20]. Dostupné z: www. ibm.com/support/knowledgecenter

[11] How Software Installation Package Managers Work On Linux. *How-To Geek* [online]. 2012 [cit. 2017-05-02]. Dostupné z: www.howtogeek.com/117579/ htg-explains-how-software-installation-package-managers-work-on-linux/

[12] Java Introduction. *W3schools* [online]. [cit. 2017-03-05]. Dostupné z: www. w3schools.in/java-tutorial/intro/

[13] HAVERBEKE, Marijn. *Eloquent javascript: a modern introduction to programming*. Second edition. San Francisco: No Starch Press, 2015. ISBN 978-1-59327-584-6.

[14] *Index of /dictionary/ITIL/en* [online]. Winfield, Illinois: Knowledge Transfer, 2011 [cit. 2017-05-05]. Dostupné z: www.knowledgetransfer.net/dictionary/ITIL/en/

[15] Logging Cheat Sheet. *OWASP* [online]. 2016 [cit. 2017-05-01]. Dostupné z: www.owasp.org/index.php/Logging_Cheat_Sheet

[16] BUCKSTEEG, Martin. *ITIL 2011*. Brno: Computer Press, 2012. ISBN 978-80-251-3732-1.

[17] *IPC incidents handling*. IBM, 2016.

[18] Top Network Monitoring Software Products. *Capterra* [online]. 2016 [cit. 2017-03-01]. Dostupné z: www.capterra.com/network-monitoring-software/

[19] Event Management: Reactive, Proactive or Predictive? *APM digest* [online]. ©2010-2017 [cit. 2017-01-21]. Dostupné z: www.apmdigest.com/event-management-reactive-proactive-or-predictive

[20] IBM AIX: An executive guide to IBM's strategy and roadmap for the AIX Operating System on Power Systems. *IBM* [online]. 2015 [cit. 2017-01-22]. Dostupné z: public.dhe.ibm.com/common/ssi/ecm/po/en/pow03169usen/POW03169USEN.PDF

[21] MILBERG, Ken. *Driving the power of AIX: performance tuning on IBM POWER systems*. Lewisville, TX: MC Press, 2009. ISBN 978-158-3470-985.

[22] Introduction to AIX. *IBM developerWorks* [online]. IBM, 2013 [cit. 2017-05-01]. Dostupné z: www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power+Systems/page/Introduction+to+AIX

[23] AIX Toolbox for Linux Applications. *IBM* [online]. [cit. 2017-01-16]. Dostupné z: www-03.ibm.com/systems/power/software/aix/linux/

[24] *IBM Support: Software lifecycle* [online]. IBM [cit. 2017-01-28]. Dostupné z: www-01.ibm.com/software/support/lifecycle/

[25] AIX Release Life Cycle View. *IBM* [online]. 2016 [cit. 2017-01-28]. Dostupné z: www.ibm.com/developerworks/community/wikis/form/anonymous/api/wiki/61ad9cf2-c6a3-4d2c-b779-61ff0266d32a/page/f1abe75a-a2b2-43dd-9d75-7dae28f5bc5f/attachment/5f3b005f-0b94-4a1c-9f26-110ae0abfdc4/media/2016-04%20AIX%20Roadmap%20and%20Lifecycle.pdf

[26] Understanding dump devices. *IBM developerWorks* [online]. 2012 [cit. 2017-01-20]. Dostupné z: www.ibm.com/developerworks/aix/library/au-aix_dump/index.html?ca=dat

[27] Managing a Dump Device. *IBM Systems Magazine* [online]. 2014 [cit. 2017-01-25]. Dostupné z: www.ibmsystemsmag.com/Blogs/AIXchange/Archive/managing-a-dump-device/

[28] X-Windows on AIX. *IBM developerWorks* [online]. 2017 [cit. 2017-01-28]. Dostupné z: www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/X-Windows%20on%20AIX

[29] Ibm aix Gallery. *Keywordsuggest.org* [online]. [cit. 2017-05-01]. Dostupné z: keywordsuggest.org/gallery/159493.html

[30] Make sudo work harder. *IBM developerWorks* [online]. 2009 [cit. 2017-01-26]. Dostupné z: www.ibm.com/developerworks/aix/library/au-sudo/

SEZNAM ZKRATEK

OS	operační systém
IS	informační systém
GUI	Graphical User Interface
JS	JavaScript
JSON	JavaScript Object Notation
RegEx	Regular Expression (někdy také RegExp)
AIX	Advanced Interactive eXecutive
TL	Technical Level
SP	Service Pack
PV	Physical Volume
PP	Physical Partition
LP	Logical Partition
LV	Logical Volume
VG	Volume Group
FS	File System
DD	Dump Device
IPC	nástroj pro automatizaci správy ICT prostředí
CMDB	Configuration management database
ROI	Return on Investment

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ